ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

ΤΜΗΜΑ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ

ΤΜΗΜΑ ΔΙΔΑΚΤΙΚΗΣ ΤΗΣ ΤΕΧΝΟΛΟΓΙΑΣ ΚΑΙ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΚΑΤΕΥΘΥΝΣΗ ΗΛΕΚΤΡΟΝΙΚΗΣ ΜΑΘΗΣΗΣ

Μεταπτυχιακή Διπλωματική Εργασία:

«Αποδοτική Διαχείριση Δεδομένων με Χρήση Πολυδιάστατων Ευρετηρίων στο Υπολογιστικό Νέφος»

Γεωργάτου Χρυσούλα

Επιβλέπων: Δουλκερίδης Χρήστος

Λέκτορας Πανεπιστημίου Πειραιώς

ΠΕΙΡΑΙΑΣ 2013

Περίληψη

Η παρούσα διπλωματική εργασία πραγματεύεται τη διαχείριση μεγάλου όγκου δεδομένων σε υπολογιστικό νέφος. Συγκεκριμένα μελετά τη δυνατότητα χρησιμοποίησης R-tree ευρετηρίων για την ταχύτερη εκτέλεση επερωτήσεων σε πολυδιάστατα δεδομένα αποθηκευμένα στο υπολογιστικό νέφος.

Αρχικά, παρουσιάζεται το υπολογιστικό νέφος και περιγράφεται η πολυδιάστατη φύση των δεδομένων καθώς και η αναγκαιότητα χρήσης ευρετηρίων κατά την εκτέλεση επερωτήσεων πάνω σε αυτά.

Στη συνέχεια περιγράφεται το framework του Hadoop και το προγραμματιστικό μοντέλο MapReduce στο οποίο και βασίζεται η προσέγγιση που παρουσιάζεται. Γίνεται μια βιβλιογραφική αναφορά αντίστοιχων προγενέστερων μελετών με στόχο τη βελτίωση του Hadoop και παρουσιάζονται τα αποτελέσματά τους. Επίσης γίνεται μια παρουσίαση της δομής ευρετηρίου R-tree και περιγράφεται η εκτέλεση επερωτήσεων με χρήση της.

Εν συνεχεία γίνεται αναφορά στη σχεδίαση που ακολουθήθηκε και στην υλοποίηση με λεπτομέρειες για τον κώδικα μου αναπτύχθηκε. Ακόλουθα περιγράφεται το περιβάλλον δοκιμών καθώς και τα σενάρια που ακολουθήθηκαν. Εκτελέστηκαν επερωτήσεις εύρους (range queries) με χρήση R-tree ευρετηρίου και χωρίς χρήση αυτού για διαφορετικά αρχεία εισόδου κατά περίπτωση, με διαφορετικό αριθμό σημείων και διαστάσεων.

Τέλος καταγράφονται τα συμπεράσματα που προέκυψαν από την πειραματική μελέτη και αποδεικνύουν ότι η χρήση R-tree ευρετηρίου βελτιώνει την απόδοση του Hadoop κατά την εκτέλεση επερωτήσεων εύρους πάνω σε πολυδιάστατα δεδομένα και ακολουθούν προτάσεις για μελλοντική έρευνα.

Λέξεις κλειδιά: Hadoop, MapReduce, R-Tree, ευρετήριο, επερώτηση εύρους, πολυδιάστατα δεδομένα, υπολογιστικό νέφος

Abstract

This master thesis deals with the management of large amounts of data in the cloud. Specifically it studies the ability of using R-tree indexes for quering multidimensional data stored in the cloud effectively.

Initially, cloud computing is presented as well as the multidimensional nature of data and the necessity of using indexes when quering multidimensional data.

In the next chapter the MapReduce programming model is presented as well as its implementation, the Hadoop framework, on which the current approach is based. Related studies are referenced followed by a presentation of the R-tree index structure and a description of range query execution over an R-tree index.

The next chapters focus on the design and the implementation of the approach including details of the code developed. Then a description of the environment, where the experimental analysis was carried out, takes place. During the experimental analysis range queries were executed, with the use of an R-tree index and without it, on different input files when it comes to the number of points includes and their dimensions.

Finally, the conclusions derived from the experimental study are demonstrated showing that using R-tree indexes indeed improves Hadoop's query performance on multidimensional data. Suggestions for future research are made.

Keywords: Hadoop, MapReduce, R-Tree, index, Range query, multidimensional data, cloud compuing

Ευχαριστίες

Στο σημείο αυτό θα ήθελα να ευχαριστήσω τον καθηγητή μου, κύριο **Δουλκερίδη Χρήστο**, για την ανάθεση της παρούσας εργασίας καθώς και για τη συμπαράσταση και τη βοήθεια που μου προσέφερε για την ολοκλήρωσή της.

Επίσης, θα ήθελα να ευχαριστήσω όλους μου τους συμφοιτητές στον 6^{ο} κύκλο του ΠΜΣ της Ηλεκτρονικής Μάθησης και ιδιαιτέρως τη συμφοιτήτριά μου και φίλη μου Αρετή Μπισιώτη για την συμπαράσταση της κατά την εκπόνηση της παρούσας εργασίας.

Τέλος, θερμές ευχαριστίες προς την οικογένειά μου και τους φίλους μου για την αμέριστη υποστήριξή τους όλα αυτά τα χρόνια. Σε εκείνους αφιερώνεται η παρούσα εργασία.

Περιεχόμενα

ΚΑΤΑΛΟΓΟΣ ΣΧΗΜΑΤΩΝ

1.ΕΙΣΑΓΩΓΗ

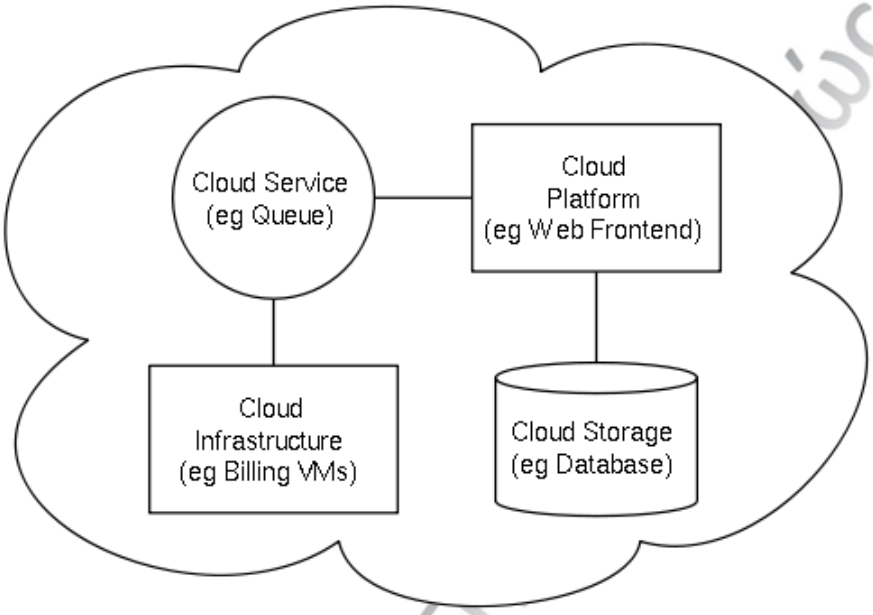
Η ηλεκτρονική μάθηση τα τελευταία χρόνια έχει γίνει αποδεκτή σαν μαθησιακό μοντέλο και είναι πλέον ιδιαίτερα δημοφιλής. Οι e-learning εφαρμογές και πλατφόρμες γνώρισαν σημαντικές και καινοτόμες αλλαγές και υπάρχει η τάση να αξιοποιηθούν οι νέες τεχνολογίες για την υποστήριξή τους. Τα υπάρχοντα μοντέλα e-learning οικοσυστημάτων υστερούν στην υποστήριξη υποκείμενων υποδομών που να κατανέμουν δυναμικά τους υπολογιστικούς και αποθηκευτικούς τους πόρους.

Το υπολογιστικό νέφος είναι ένα ιδιαίτερα δημοφιλές προγραμματιστικό πρότυπο και είναι η ελπιδοφόρα υποδομή που θα μπορούσε να προσδώσει ιδιαίτερη αξία στα e-learning οικοσυστήματα εξαιτίας της δυνατότητάς του να προσφέρει πόρους υπολογισμού και αποθήκευσης σαν υπηρεσίες (services) [1]. Η τεχνολογία του υπολογιστικού νέφους και ιδιαίτερα η επεξεργασία δεδομένων με αυτό θα μελετηθεί στην παρούσα εργασία.

1.1 ΥΠΟΛΟΓΙΣΤΙΚΟ ΝΕΦΟΣ

Στη σημερινή εποχή ένας αυξανόμενος αριθμός εταιριών πρέπει να επεξεργαστεί τεράστιο όγκο δεδομένων με τρόπο αποτελεσματικό από άποψη κόστους. Χαρακτηριστικά παραδείγματα τέτοιων εταιριών είναι εταιρίες όπως η Microsoft, η Google, η Facebook και η Yahoo. Ο όγκος των δεδομένων που καλούνται να επεξεργαστούν καθημερινά καθιστά τις ήδη υπάρχουσες λύσεις βάσεων δεδομένων μη αποδοτικές [2]. Αντί αυτού τέτοιες εταιρίες έχουν διαδώσει ένα αρχιτεκτονικό πρότυπο που βασίζεται σε μεγάλο αριθμό commodity servers. Για εταιρίες που επεξεργάζονται μεγάλο όγκο δεδομένων μόνο περιστασιακά το να διατηρούν δικό τους κέντρο δεδομένων δεν είναι εφικτό. Αντί αυτού το υπολογιστικό νέφος εμφανίζεται σαν μια ελπιδοφόρα προσέγγιση για ενοικίαση μεγάλης υποδομής πληροφορικής για μικρή διάρκεια επί πληρωμή [12]. Στα συστήματα νέφους ένας πάροχος υπηρεσιών προσφέρει ελαστικούς υπολογιστικούς πόρους σε έναν αριθμό χρηστών οι οποίοι δε γνωρίζουν λεπτομέρειες για την υποκείμενη υποδομή [14]. Το υπολογιστικό νέφος είναι ένα πολύ επιτυχημένο παράδειγμα υπολογισμού προσανατολισμένου σε υπηρεσίες και έχει αλλάξει ριζικά τον τρόπο που οι υπολογιστικές υποδομές οργανώνονται και χρησιμοποιούνται [3].

Ένα νέφος είναι ένας τύπος διαμοιραζόμενου κέντρου δεδομένων που παρέχει τις υποδομές σαν υπηρεσίες. Αποτελείται από ογκώδεις πόρους και παρουσιάζεται σαν ένα η περισσότερους ενοποιημένους φορείς πόρων που προσφέρουν μέσω υπηρεσιών σε χρήστες / εφαρμογές τη δυνατότητα να χρησιμοποιήσουν αυτούς τους πόρους χωρίς να έχουν λεπτομερείς πληροφορίες. Μία από τις βασικότερες ιδέες πίσω από το υπολογιστικό νέφος είναι η επεκτασιμότητα και η τεχνολογία που την κάνει πιθανή είναι το virtualization [1].



Εικόνα 1: Αρχιτεκτονική του υπολογιστικού νέφους [34]

Προβλήματα όπως η επεξεργασία crawled documents ή η επαναδημιουργία ενός web ευρετηρίου σπάνε σε πολλές ανεξάρτητες υπό-εργασίες οι οποίες μοιράζονται στους διαθέσιμους κόμβους και υπολογίζονται παράλληλα. Για την απλοποίηση της ανάπτυξης διαμοιραζόμενων εφαρμογών πάνω σε αυτή την αρχιτεκτονική έχουν επίσης προχωρήσει σε αντίστοιχες υλοποιήσεις επεξεργασίας δεδομένων. Χαρακτηριστικά παραδείγματα τέτοιων υλοποιήσεων είναι το MapReduce της Google, το Dryad της Microsoft και το Map –Reduce-Merge της Yahoo [2]. Παρότι τα συστήματα αυτά έχουν σχεδιαστικές διαφορές τα προγραμματιστικά τους μοντέλα μοιράζονται κοινούς στόχους με βασικότερους την απόκρυψη των λεπτομερειών της παράλληλης επεξεργασίας, την αντοχή σε σφάλματα και την βελτιστοποίηση στην εκτέλεση. Οι προγραμματιστές συνεχίζουν να γράφουν σειριακά

προγράμματα και το πλαίσιο (framework) αναλαμβάνει να διαμοιράσει το πρόγραμμα ανάμεσα στους διαθέσιμους κόμβους και να εκτελέσει κάθε instance του προγράμματος στο κατάλληλο τμήμα δεδομένων [2].

1.2 ΠΟΛΥΔΙΑΣΤΑΤΑ ΔΕΔΟΜΕΝΑ ΚΑΙ ΕΠΕΡΩΤΗΣΕΙΣ

Στη σημερινή εποχή τα δεδομένα τείνουν να γίνουν εξαιρετικά πολύπλοκα ενώ πολλά από τα δεδομένα που οι εταιρίες καλούνται να επεξεργαστούν πλέον είναι πολυδιάστατα. Για παράδειγμα τα Location-Based Services χρησιμοποιούν πολυδιάστατα δεδομένα και συναντώνται σε πολλές πραγματικές εφαρμογές όπως είναι το Facebook, το Flickr, το Twitter κ.α. [4].

Στον πραγματικό κόσμο οι χρήστες τείνουν να πραγματοποιούν επερωτήσεις με περισσότερα από ένα κλειδιά. Για παράδειγμα σε ένα διαδικτυακό σύστημα βίντεο όπως είναι το YouTube κάθε βίντεο αποθηκεύεται σε ένα αποθηκευτικό χώρο κλειδιού-τιμής με κλειδί μια μοναδική ταυτότητα βίντεο και τιμή τις πληροφορίες του βίντεο που περιλαμβάνουν τον τίτλο, την ώρα που ανέβηκε το βίντεο και τον αριθμό από views. Παρότι το βίντεο μπορεί λογικά να ανακτηθεί από την ταυτότητα ένα σύνηθες σενάριο είναι ένας χρήστης να θελήσει να βρει βίντεο με συγκεκριμένους τίτλους ή σε ένα εύρος ημερομηνιών [5]. Ακόμα και στην ηλεκτρονική μάθηση τα elearning αντικείμενα (objects) μπορούν να αναπαρασταθούν με ένα σύνολο μεταδεδομένων στα οποία ο χρήστης μπορεί να πραγματοποιήσει επερωτήσεις.

Η επεξεργασία τέτοιων δεδομένων είναι αρκετά δύσκολη και πολύπλοκη καθώς δεν υπάρχει περιορισμός στις διαστάσεις των σημείων ενώ οι επερωτήσεις που χρειάζεται να εκτελεστούν πάνω στα πολυδιάστατα αυτά δεδομένα είναι συχνά επιλεκτικές και δεν απαιτούν πρόσβαση στο σύνολό τους αλλά μόνο σε μέρος αυτών. Χαρακτηριστικά πολυδιάστατα δεδομένα είναι τα χωρικά δεδομένα και οι επερωτήσεις που εκτελούνται σε αυτά όπως οι επερωτήσεις εύρους που θα παρουσιαστούν στη συνέχεια τις περισσότερες φορές απαιτούν πρόσβαση σε ένα πεπερασμένο υποσύνολο των δεδομένων αυτών και όχι στο σύνολό τους.

1.3 ΔΙΑΤΥΠΩΣΗ ΠΡΟΒΛΗΜΑΤΟΣ

Το υπολογιστικό νέφος και τα πλαίσια όπως το MapReduce και το Hadoop είναι αποδεδειγμένα κατάλληλα για επεξεργασία πολύ μεγάλου όγκου δεδομένων. Επίσης όμως έχουν καταγραφεί σημαντικά μειονεκτήματα στην απόδοση τους κατά την εκτέλεση επερωτήσεων σε σχέση με τα ΣΔΒΔ λόγω της απουσίας ευρετηρίων [6] [7]. Ειδικά αν οι επερωτήσεις έχουν υψηλή επιλεκτικότητα είναι σημαντικό να μην απαιτείται πρόσβαση στο σύνολο των δεδομένων παρά μόνο σε μέρος αυτών. Στη σημερινή εποχή που το μεγαλύτερο πλήθος των δεδομένων είναι πολυδιάστατα αποτελεί πρόκληση η διαχείριση πολυδιάστατων δεδομένων σε υπολογιστικό νέφος καθώς και η βελτίωση της απόδοσης εκτέλεσης επερωτήσεων σε πολυδιάστατα δεδομένα πάνω σε υπολογιστικό νέφος με την αξιοποίηση ευρετηρίων. Στόχοι της παρούσας εργασίας είναι

- Να ενσωματωθεί η δυνατότητα χρήσης ευρετηρίων στο MapReduce, με τρόπο κατά το δυνατόν διάφανο για τις εφαρμογές, για τη βελτίωση της απόδοσης MapReduce εργασιών
- Να αξιολογηθεί ο βαθμός στον οποίο η χρήση ευρετηρίων βελτιώνει την απόδοση της εκτέλεσης επερωτήσεων και το ποσοστό μείωσης του αριθμού των δεδομένων που θα πρέπει να επεξεργαστούν.

1.4 ΔΙΑΡΘΡΩΣΗ ΕΡΓΑΣΙΑΣ

Η διάρθρωση της παρούσας εργασίας παρουσιάζεται στη συνέχεια.

Κεφάλαιο 2: στο κεφάλαιο αυτό αναλύονται τα προαπαιτούμενα για το MapReduce και για το Hadoop που αποτελούν το υποκείμενο επίπεδο κατά την εκτέλεση των επερωτήσεων καθώς και τα μειονεκτήματά τους που έχουν παρατηρηθεί.

Κεφάλαιο 3: στο κεφάλαιο αυτό γίνεται μια βιβλιογραφική επισκόπηση παλαιότερων ερευνών και παρουσιάζονται προσπάθειες βελτίωσης του Hadoop.

Κεφάλαιο 4: στο κεφάλαιο αυτό παρουσιάζονται τα προαπαιτούμενα για τα R-trees που είναι η δομή ευρετηρίου που επιλέχθηκε για τη συγκεκριμένη υλοποίηση και γίνεται μια περιγραφή της εκτέλεσης των επερωτήσεων εύρους και κοντινότερου γείτονα με χρήση R- δέντρου.

Κεφάλαιο 5: στο κεφάλαιο αυτό παρουσιάζεται το πρόβλημα στο οποίο η παρούσα μελέτη καλείται να απαντήσει.

Κεφάλαιο 6: στο κεφάλαιο αυτό παρουσιάζεται η σχεδίαση που ακολουθήθηκε στην προσέγγιση.

Κεφάλαιο 7: στο κεφάλαιο αυτό παρουσιάζεται η υλοποίηση της προσέγγισης με λεπτομέρειες για τις κλάσεις που αναπτύχθηκαν.

Κεφάλαιο 8: στο κεφάλαιο αυτό παρουσιάζεται η πειραματική ανάλυση με λεπτομέρειες για το περιβάλλον και τις συνθήκες δοκιμών καθώς και αναλυτικές μετρήσεις

Κεφάλαιο 9: στο κεφάλαιο αυτό αναλύονται τα συμπεράσματα όπως προέκυψαν από την πειραματική μελέτη και γίνονται προτάσεις για μελλοντική έρευνα.

2.HADOOP - MAPREDUCE

2.1 MAPREDUCE

Το MapReduce είναι ένα προγραμματιστικό μοντέλο και η αντίστοιχη υλοποίηση για την επεξεργασία και τη δημιουργία μεγάλων σετς δεδομένων που αναπτύχθηκε στη Google από τους Jeffrey Dean και Sanjay Ghemawat [8] [9]. Στόχος του να επιτρέπει στους προγραμματιστές να επικεντρωθούν σε απλούς υπολογισμούς και να μην αναλώνονται στις λεπτομέρειες της παράλληλης εκτέλεσης, της αντοχής σε σφάλματα, του διαμοιρασμού των δεδομένων και της εξισορρόπησης του φορτίου. Η ιδέα για την ανάπτυξη του προήλθε από τα map - reduce όπως αυτά προϋπήρχαν στη Lisp και σε άλλες λειτουργικές γλώσσες προγραμματισμού [6].

Για τη δημιουργία μιας map/reduce εργασίας ο χρήστης προσδιορίζει μια συνάρτηση map η οποία επεξεργάζεται ένα ζεύγος κλειδιού/τιμής για να δημιουργήσει ένα σετ από ενδιάμεσα ζεύγη κλειδιού/τιμής καθώς και μια συνάρτηση reduce η οποία συγχωνεύει όλες τις ενδιάμεσες τιμές που σχετίζονται με το ίδιο ενδιάμεσο κλειδί [10] [11] [15]. Οι συναρτήσεις map reduce αποτυπώνονται ακόλουθα:

map (k1,v1) → list(k2,v2)
reduce (k2,list(v2)) → list(v2) [12]

Τυπικά μόνο κανένα ή ένα αποτέλεσμα τιμών προκύπτει από τη συνάρτηση reduce. Η συνάρτηση reduce τροφοδοτείται με τις ενδιάμεσες τιμές από έναν iterator**.** Με τον τρόπο αυτό μπορεί κανείς να διαχειριστεί λίστες οι οποίες είναι πολύ μεγάλες για να χωρέσουν στην μνήμη [8].

Αναλυτικά η επεξεργασία των δεδομένων γίνεται μέσω των ακόλουθων 6 βημάτων [7]:

1. Input reader: ο Input reader στη βασική του μορφή παίρνει είσοδο από αρχεία και τη μετατρέπει σε ζεύγη κλειδιού/τιμή. Τα δεδομένα χωρίζονται σε splits που είναι η μονάδα δεδομένων που επεξεργάζεται το map task
2. Συνάρτηση map: Η συνάρτηση map παίρνει σαν είσοδο ένα ζεύγος κλειδιού τιμής από τον input reader, εκτελεί σε αυτό τη λογική της συνάρτησης map και παράγει ένα νέο ζεύγος κλειδιού / τιμής. Τα αποτελέσματα του map task αρχικά αποθηκεύονται σε έναν buffer της κύριας μνήμης και όταν γεμίσει σκορπίζονται στο δίσκο. Τα σκορπισμένα αρχεία στο τέλος συγχωνεύονται σε ένα ταξινομημένο αρχείο

3. Συνάρτηση Combiner: Η συνάρτηση αυτή είναι προαιρετική και παρέχεται για την περίπτωση που υπάρχει σημαντική επανάληψη στα ενδιάμεσα κλειδιά που παράγονται από τα map tasks και η ορισμένη συνάρτηση Reduce είναι αντιμεταθετική και συσχετιστική. Σε αυτή την περίπτωση η συνάρτηση Combiner εκτελεί μερική μείωση ώστε τα ζεύγη με κοινό κλειδί να επεξεργαστούν σα μια ομάδα από το reduce task.
4. Συνάρτηση Partition: Μια συνάρτηση κατακερματισμού χρησιμοποιείται σαν προεπιλογή για να διαχωρίσει τα ενδιάμεσα κλειδιά που περνούν από το map task στο reduce task. Παρότι σε γενικές γραμμές η χρήση της συνάρτησης αυτής εξασφαλίζει καλή εξισορρόπηση σε αρκετές περιπτώσεις κρίνεται σκόπιμο να χρησιμοποιηθούν άλλες συναρτήσεις που ορίζονται από το χρήστη.
5. Συνάρτηση Reduce: Η συνάρτηση Reduce καλείται μια φορά για κάθε διακριτό κλειδί και εφαρμόζεται στο σύνολο των συσχετιζόμενων με αυτό το κλειδί τιμών. Η είσοδος σε κάθε reduce task είναι εγγυημένο ότι θα επεξεργαστεί σε αύξουσα σειρά κλειδιού. Είναι δυνατό να οριστεί από το χρήστη μια συνάρτηση σύγκρισης κατά τη διαδικασία ταξινόμησης.
6. Output writer: Ο output writer είναι υπεύθυνος να γράψει το αποτέλεσμα σε ένα σταθερό αποθηκευτικό χώρο, συνήθως αρχείο αλλά υπάρχει η δυνατότητα η συνάρτηση αυτή να τροποποιηθεί.

Το MapReduce χρησιμοποιεί το Google File System(GFS) σαν ένα υποκείμενο επίπεδο αποθήκευσης για να διαβάζει τα δεδομένα εισόδου και να αποθηκεύει τα δεδομένα εξόδου. Το GFS είναι ένα διαμοιραζόμενο σύστημα αρχείων το οποίο υποστηρίζει αντοχή στα σφάλματα μέσω του διαχωρισμού και της αντιγραφής των δεδομένων.

Τα προγράμματα που γράφονται ακολουθώντας αυτό το λειτουργικό μοντέλο αυτόματα παραλληλοποιούνται και εκτελούνται σε μεγάλα νέφη από «φτηνά» μηχανήματα. Το runtime αναλαμβάνει τις λεπτομέρειες για το διαχωρισμό των δεδομένων εισόδου, τον προγραμματισμό της εκτέλεσης του προγράμματος σε ένα σετ από μηχανήματα, το χειρισμό των αποτυχιών των μηχανημάτων και τη διαχείριση της επικοινωνίας ανάμεσα στις μηχανές. Σαν αποτέλεσμα προγραμματιστές χωρίς αντίστοιχη εμπειρία με παράλληλα και κατανεμημένα συστήματα έχουν τη δυνατότητα να αξιοποιήσουν τους πόρους τους,

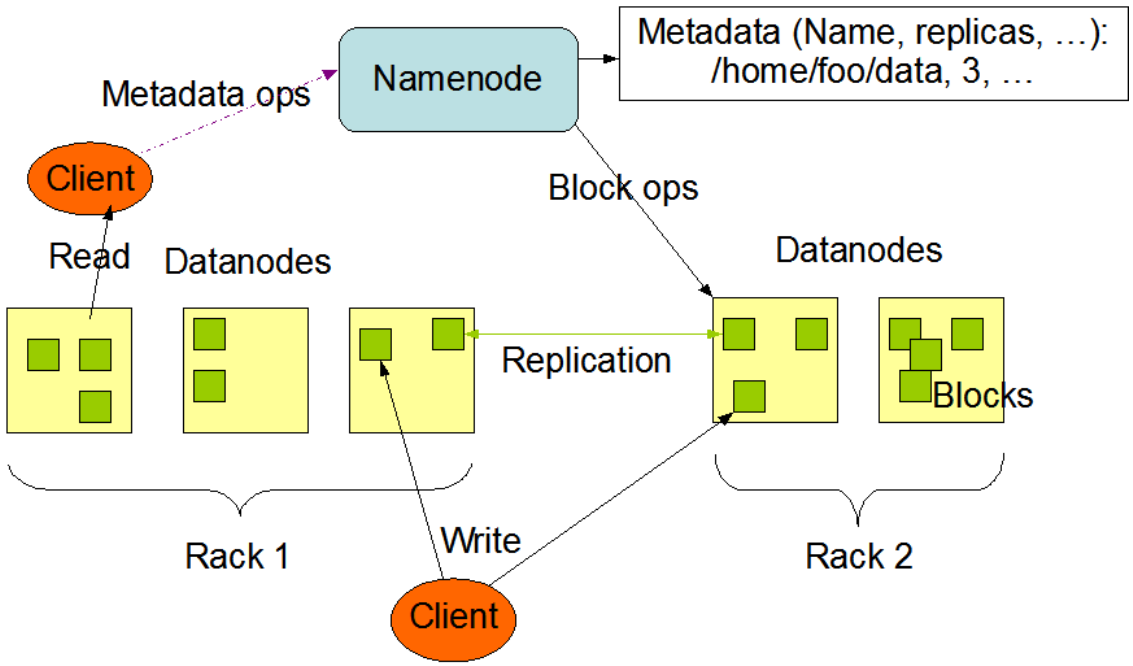
2.2 HADOOP

Το Apache Hadoop είναι μια open source υλοποίηση του MapReduce γραμμένη σε Java και η πιο δημοφιλής. Το Hadoop όπως και το MapReduce αποτελείται από δύο επίπεδα: ένα επίπεδο για την αποθήκευση των δεδομένων - το Hadoop dfs (hdfs) και ένα επίπεδο επεξεργασίας των δεδομένων που ονομάζεται Hadoop MapReduce Framework [20].

2.2.1 HDFS

Το HDFS έχει master/slave αρχιτεκτονική. Μια συστάδα HDFS αποτελείται από έναν μόνο NameNode, έναν master server που διαχειρίζεται το namespace του συστήματος αρχείων και ρυθμίζει την πρόσβαση των πελατών στα αρχεία. Συμπληρωματικά υπάρχει ένας αριθμός DataNodes, συνήθως ένας ανά κάθε κόμβο της συστάδας, που ελέγχουν την αποθήκευση στον κόμβο πάνω στον οποίο τρέχουν [20]. Το HDFS εκθέτει ένα file system namespace και επιτρέπει στα δεδομένα των χρηστών να αποθηκευτούν σε αρχεία. Εσωτερικά κάθε αρχείο σπάει σε ένα ή περισσότερα μπλοκς τα οποία με τη σειρά τους αποθηκεύονται στους datanodes. Ο NameNode εκτελεί εργασίες του namespace του συστήματος αρχείων όπως άνοιγμα, κλείσιμο και μετονομασία αρχείων και φακέλων. Επιπρόσθετα καθορίζει την αντιστοίχιση των μπλοκς στους datanodes. Οι DataNodes είναι υπεύθυνοι για την εξυπηρέτηση αιτημάτων ανάγνωσης και εγγραφής από τους πελάτες του συστήματος αρχείων. Επιπλέον δημιουργούν, διαγράφουν και αντιγράφουν μπλοκς μετά από οδηγίες του NameNode [10][21].

Η ύπαρξη ενός και μοναδικού NameNode στη συστάδα απλοποιεί σημαντικά την αρχιτεκτονική του συστήματος. Ο NameNode κρατά και ελέγχει όλα τα μεταδεδομένα του HDFS. Με τον τρόπο αυτό τα μεταδεδομένα είναι αποκομμένα από τα δεδομένα του συστήματος το οποίο έχει σχεδιαστεί με τέτοιο τρόπο ώστε τα δεδομένα των χρηστών να μην περνάνε πότε μέσα από το NameNode.



Εικόνα 2: Αρχιτεκτονική HDFS [20]

Μια πλήρως διαμορφωμένη HDFS συστάδα τρέχει ένα σετ από daemons που περιλαμβάνουν [22]:

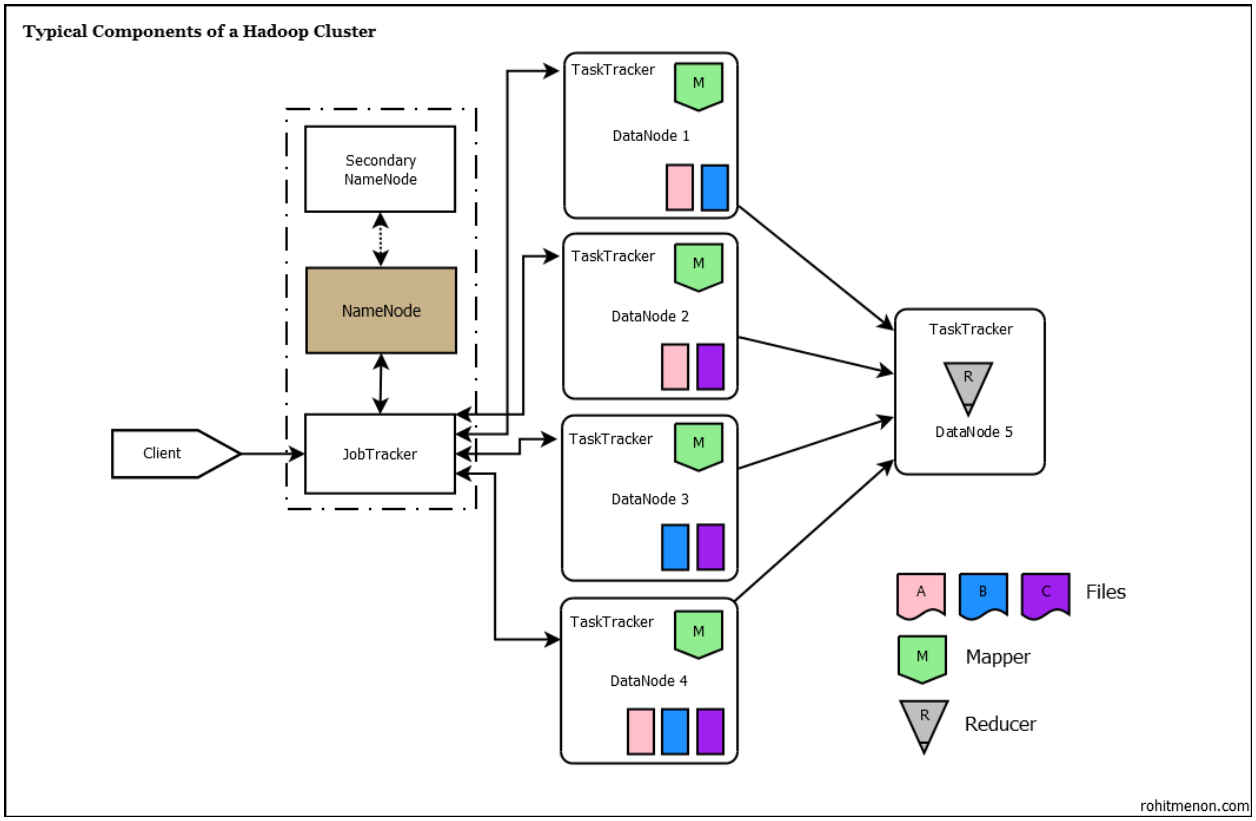
1. **NameNode** : Είναι ο master του HDFS ο οποίος διευθύνει τους σκλάβους DataNode daemons.

2. **DataNode** : είναι ο εργάτης του κατανεμημένου συστήματος αρχείων ο οποίος γράφει και διαβάζει HDFS μπλοκς σε πραγματικά αρχεία στο τοπικό σύστημα αρχείων.

3. **SecondaryNameNode** : Είναι ένας βοηθητικός daemon για την παρακολούθηση της κατάστασης της HDFS συστάδας.

4. **JobTracker** : Είναι ο σύνδεσμος ανάμεσα στις εφαρμογές-πελάτες και στο Hadoop, αποφασίζει το πλάνο εκτέλεσης για τις καταχωρημένες εργασίες, αναθέτει διαφορετικές εργασίες στους κόμβους και παρακολουθεί όλες τις εργασίες που τρέχουν.

5. **TaskTracker** : Είναι υπεύθυνος για την εκτέλεση των μεμονωμένων εργασιών που αναθέτει ο JobTracker.

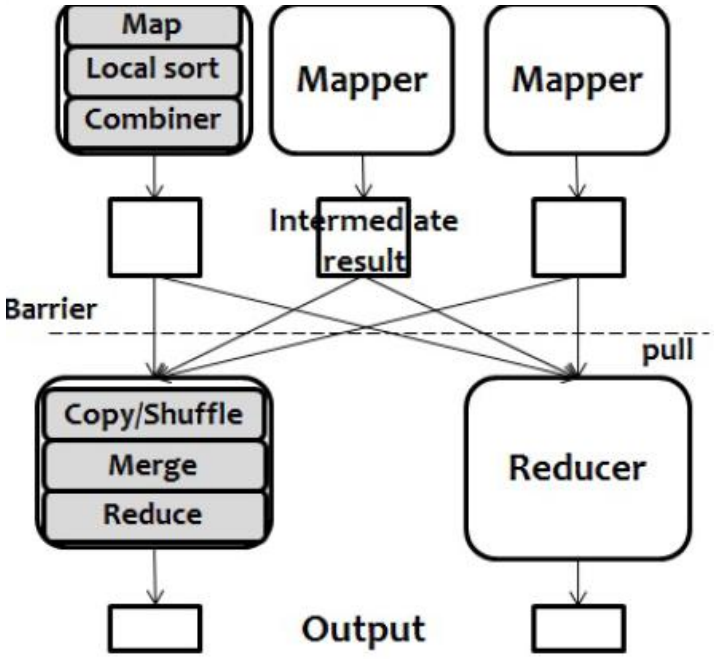


Εικόνα 3: Συστάδα Hadoop [22]

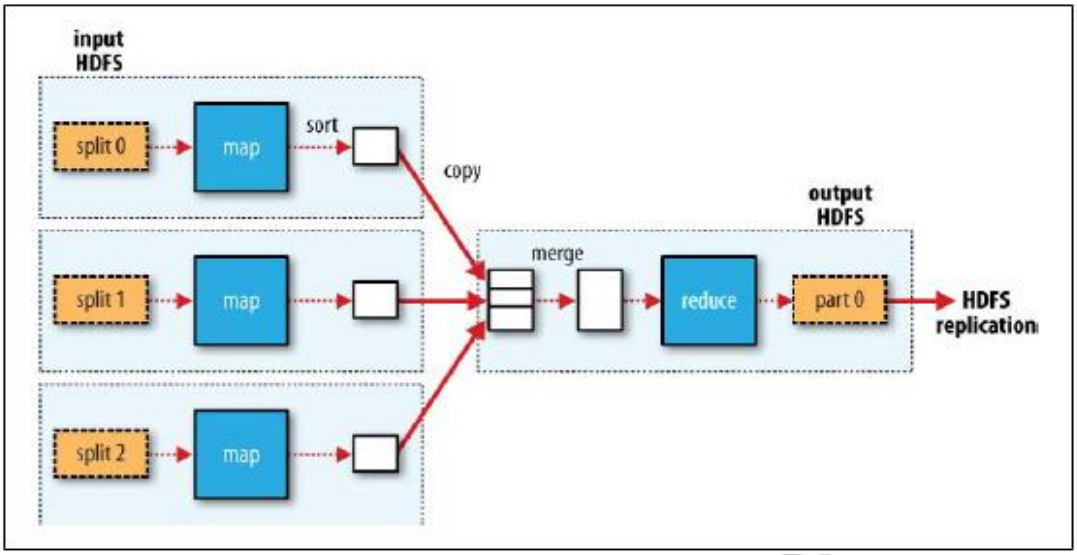
2.2.2 MAPREDUCE FRAMEWORK

Μία εργασία MapReduce χωρίζεται σε δύο στάδια, ένα map και ένα reduce [10] [11]. Ο master κόμβος βρίσκει ελεύθερους κόμβους – εργάτες και τους αναθέτει από ένα map ή reduce task ανάλογα με το στάδιο. Πριν ξεκινήσει η εργασία map ένα αρχείο εισόδου φορτώνεται στο hdfs. Καθώς φορτώνεται σπάει σε πολλαπλά μπλοκ δεδομένων που έχουν το ίδιο μέγεθος και κάθε μπλοκ αντιγράφεται 3 φορές εξασφαλίζοντας έτσι αντοχή σε σφάλματα. Στη συνέχεια κάθε μπλοκ ανατίθεται σε ένα mapper (κόμβος εργάτης στον οποίο έχει ανατεθεί ένα map task) που εφαρμόζει τη συνάρτηση Map σε κάθε εγγραφή του μπλοκ δεδομένων. Τα ενδιάμεσα αποτελέσματα που προέκυψαν από τους mappers ταξινομούνται τοπικά για να ομαδοποιηθούν τα ζεύγη κλειδιού-τιμής που μοιράζονται κοινό κλειδί. Μετά από την τοπική ταξινόμηση η συνάρτηση combine προαιρετικά εφαρμόζεται για να βρει ένα πρώιμο άθροισμα στα ομαδοποιημένα ζεύγη κλειδιού – τιμής ώστε να ελαχιστοποιηθεί το κόστος της μεταφοράς όλων

των ενδιάμεσων αποτελεσμάτων στο reducer. Στη συνέχεια τα mapped αποτελέσματα αποθηκεύονται στους τοπικούς δίσκους στους mappers και χωρίζονται σε τόσα τμήματα όσοι και οι reducers της MapReduce εργασίας. Όταν ολοκληρωθούν όλα τα map tasks ανατίθενται reduce tasks στους εργάτες κόμβους. Επειδή όλα τα mapped αποτελέσματα είναι ήδη χωρισμένα και αποθηκευμένα σε τοπικούς δίσκους κάθε reducer πραγματοποιεί το ανακάτεμα (shuffling) απλά τραβώντας το τμήμα(partition) των mapped αποτελεσμάτων που του αντιστοιχεί από τους mappers. Ένας reducer διαβάζει τα ενδιάμεσα αποτελέσματα και τα συγχωνεύει με βάση τα ενδιάμεσα κλειδιά ώστε όλες οι τιμές με το ίδιο κλειδί να ομαδοποιηθούν. Στη συνέχεια κάθε reducer εφαρμόζει τη συνάρτηση Reduce στις ενδιάμεσες τιμές για κάθε κλειδί που συναντά. Τα αποτελέσματα των reducers αποθηκεύονται και αντιγράφονται 3 φορές στο hdfs [20].



Εικόνα 4: Αρχιτεκτονική Hadoop [6]



Εικόνα 5: Ροή δεδομένων κατά τη MapReduce εργασία [10]

Το MapReduce framework εκτελεί όλες τις εργασίες βασιζόμενο σε ένα σχήμα προγραμματισμού που διαμορφώνεται στο χρόνο εκτέλεσης . Σε αντίθεση με τα ΣΔΒΔ το MapReduce framework δεν καθορίζει ποιες εργασίες θα τρέξουν σε ποιους κόμβους πριν την εκτέλεση αλλά το αποφασίζει ολοκληρωτικά στο χρόνο εκτέλεσης (runtime) και επιτυγχάνει με αυτόν τον τρόπο αντοχή στα σφάλματα εντοπίζοντας τις αποτυχίες και αναθέτοντας τις εργασίες των αποτυχημένων κόμβων σε υγιείς. Σε κόμβους που ολοκλήρωσαν τις εργασίες τους αναθέτονται νέα μπλοκς εισόδου. Με τον τρόπο αυτό επιτυγχάνεται εξισορρόπηση του φόρτου με τους γρηγορότερους κόμβους να επεξεργάζονται περισσότερα δεδομένα εισόδου και τις εργασίες κόμβων που αντιμετωπίζουν δυσκολίες να αναθέτονται σε ελεύθερους κόμβους που ολοκλήρωσαν τις δικές τους εργασίες.

Τέλος οι εργασίες map reduce εκτελούνται χωρίς να απαιτείται επικοινωνία με άλλες εργασίες και έτσι δε δημιουργείται καμία διένεξη στο συγχρονισμό και κανένα κόστος επικοινωνίας ανάμεσα σε εργασίες κατά τη διάρκεια μιας εκτέλεσης MR.

2.3 ΚΡΙΤΙΚΗ

Το MapReduce και ιδιαίτερα το Hadoop, κέρδισε έδαφος και θεωρήθηκε το επόμενο βήμα στη διαχείριση μεγάλου όγκου δεδομένων [13], παράλληλα όμως δέχτηκε και έντονη κριτική καθώς θεωρήθηκε ότι αποτελεί ένα βήμα πίσω στην παράλληλη επεξεργασία όπως αυτή επιτυγχάνεται στα ΣΔΒΔ [14] [13] [16] [19]. Το MapReduce υστερεί σε πολλά από τα χαρακτηριστικά [15] [24] [7] που έχουν αποδειχτεί πολύτιμα για την ανάλυση δομημένων δεδομένων κυρίως γιατί αρχικά δε σχεδιάστηκε για να πραγματοποιεί ανάλυση δομημένων δεδομένων.

Έρευνες που πραγματοποιήθηκαν έδειξαν μια καθαρή ανταλλαγή ανάμεσα στην απόδοση και την ανοχή σε σφάλματα. Το MapReduce αυξάνει την ανοχή σε σφάλματα μέσω της ανάλυσης με συχνούς ελέγχους των ολοκληρωμένων εργασιών και της αντιγραφής των δεδομένων. Οι συχνοί αυτοί όμως έλεγχοι έχουν τίμημα στην αποδοτικότητα. Από τη μεριά τους τα ΣΔΒΔ στοχεύουν στην απόδοση περισσότερο παρά στην αντοχή στα σφάλματα. Τα ΣΔΒΔ εκμεταλλεύονται τη σύνδεση ενδιάμεσων αποτελεσμάτων ανάμεσα στους εκτελεστές επερωτήσεων με πιθανό κίνδυνο όμως να απαιτηθεί μεγάλος αριθμός επανάληψης εργασιών σε περίπτωση αποτυχίας.

Τα πλεονεκτήματα του Hadoop συνοψίζονται στα εξής [6]:

- Είναι απλό και εύκολο στη χρήση καθώς ο προγραμματιστής χρειάζεται να ορίσει μόνο συναρτήσεις map – reduce
- Είναι ευέλικτο καθώς δεν εξαρτάται από κανένα μοντέλο ή σχήμα και ο προγραμματιστής μπορεί να διαχειριστεί αδόμητα δεδομένα ευκολότερα από ότι με τα ΣΔΒΔ
- Είναι ανεξάρτητο από το υποκείμενο επίπεδο μνήμης – αποθήκευσης και δυνατότητα να συνεργαστεί με διαφορετικούς τύπους.
- Παρέχει αντοχή στα σφάλματα
- Προσφέρει υψηλή επεκτασιμότητα

Αντίστοιχα παρουσιάζει και σημαντικά μειονεκτήματα σε σύγκριση με τα ΣΔΒΔ και συχνά παρουσιάζεται σαν ένα Extract-Transform-Load(ETL) εργαλείο [13]. Τα σημαντικότερα μειονεκτήματα του συνοψίζονται στα εξής [6]:

- Δεν υποστηρίζει καμία γλώσσα υψηλού επιπέδου και καμία τεχνική βελτιστοποίησης επερωτήσεων
- Δε διαθέτει σχήμα ή ευρετήριο (index) με αποτέλεσμα μην εκμεταλλεύεται τα πλεονεκτήματα της μοντελοποίησης δεδομένων και να προκαλεί πτώση στην απόδοση
- Δυσκολία να χρησιμοποιηθεί για πολύπλοκους αλγορίθμους μόνο με χρήση MR εργασιών
- Χαμηλή αποδοτικότητα καθώς δίνει πρωτεύουσα σημασία στην αντοχή στα σφάλματα και στην επεκτασιμότητα οι λειτουργίες του δεν είναι βελτιστοποιημένες για αποδοτικότητα I/O
- Είναι πολύ νέο σε σχέση με τα ΣΔΒΔ

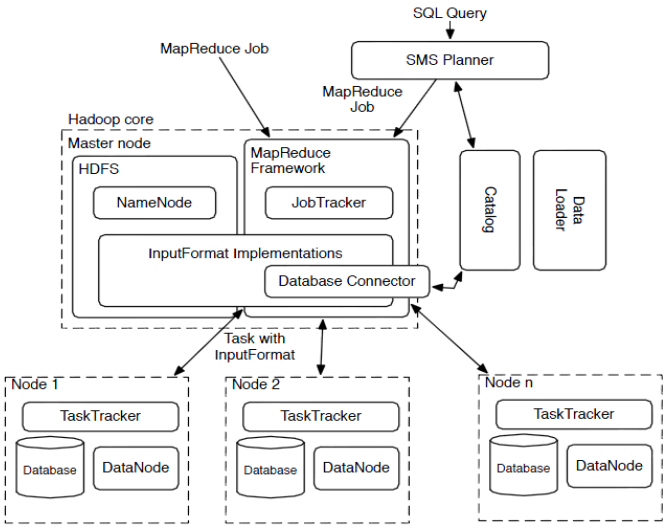
Η παρούσα διπλωματική εργασία επικεντρώνεται στο δεύτερο από τα προαναφερθέντα μειονεκτήματα που αφορά την απουσία ευρετηρίων και επιχειρεί μια βελτίωση τα απόδοσης του Hadoop με χρήση τους. Προτείνεται μια προσέγγιση με χρήση δομής πολυδιάστατου ευρετηρίου κατά την εκτέλεση επερωτήσεων όπως οι επερωτήσεις εύρους. Η προσέγγιση αυτή στοχεύει στη μείωση του ποσοστού των δεδομένων στα οποία θα πρέπει να έχει πρόσβαση η MapReduce εργασία κατά την επεξεργασία της επερώτησης, ιδιαίτερα όταν δεν απαιτείται εξαντλητική πρόσβαση στο σύνολο τους καθώς και μια βελτίωση στο χρόνο εκτέλεσης των επερωτήσεων.

3. ΒΕΛΤΙΩΣΕΙΣ –RELATED WORK

Τα μειονεκτήματα του Hadoop σε σχέση με τα ΣΔΒΔ οδήγησαν σε προσπάθειες βελτίωσης της απόδοσης με υβριδικά συστήματα ή βελτιώσεις πάνω από το υπάρχον framework [17] [18] [33], οι σημαντικότερες των οποίων παρουσιάζονται παρακάτω.

3.1 HADOOP DB

Οι A.Abouzeid et. al παρουσίασαν το HadoopDB, ένα υβριδικό σύστημα που συνδυάζει τα πλεονεκτήματα του MapReduce και των ΣΔΒΔ [24]. Η βασική ιδέα πίσω από το HadoopDB είναι η σύνδεση πολλαπλών ΣΔΒΔ χρησιμοποιώντας το Hadoop σαν επίπεδο συνεργασίας των εργασιών και επικοινωνίας του δικτύου. Οι επερωτήσεις παραλληλίζονται στους κόμβους χρησιμοποιώντας το Hadoop framework ενώ όσο περισσότερη εργασία μπορεί να γίνει για τις επερωτήσεις σε ένα κόμβο προωθείται στους αντίστοιχους κόμβους – βάσεις δεδομένων. Επιτυγχάνεται ανοχή στα σφάλματα και ικανότητα λειτουργίας σε ετερογενή περιβάλλοντα με την κληρονόμηση της υλοποίησης του Hadoop για τον προγραμματισμό και την παρακολούθηση των εργασιών και επιπλέον επιτυγχάνεται η απόδοση των παράλληλων συστημάτων βάσεων με την επεξεργασία μεγάλου τμήματος των επερωτήσεων μέσα στη database engine.



Εικόνα 6: Αρχιτεκτονική του HadoopDB [24]

Το Hadoop DB επεκτείνει το Hadoop με τα εξής τέσσερα components.

Database Connector: είναι η διασύνδεση ανάμεσα σε ανεξάρτητα συστήματα βάσεων δεδομένων που βρίσκονται σε κόμβους της συστάδας και σε Task trackers. Επεκτείνει την κλάση του Hadoop Input Format και είναι μέρος της βιβλιοθήκης Input- Format Implementations. Κάθε εργασία map-reduce προμηθεύει τον connector με μια SQL επερώτηση και παραμέτρους σύνδεσης. Ο connector συνδέεται στη βάση, εκτελεί την επερώτηση και επιστρέφει ζεύγη κλειδιού τιμής.

Catalog: διατηρεί μετά-πληροφορίες για τις βάσεις δεδομένων όπως παραμέτρους σύνδεσης και μετά-δεδομένα όπως τα σετ δεδομένων που περιλαμβάνονται στη συστάδα, τις τοποθεσίες των αντιγράφων ασφαλείας και ιδιότητες διαχωρισμού δεδομένων.

Data Loader : είναι υπεύθυνος για α) τον καθολικό επαναμερισμό των δεδομένων για συγκεκριμένο κλειδί κατά τη φόρτωση β)για το χωρισμό δεδομένων ενός κόμβου σε πολλαπλά μικρότερα μέρη – chunks και γ) τελικά τη φόρτωση των βάσεων δεδομένων ενός κόμβου με τα chunks. Αποτελείται από δύο μέρη έναν Global Hasher και ένα Local Hasher. Ο Global Hasher εκτελεί μια ειδικά σχεδιασμένη map reduce εργασία πάνω από το Hadoop το οποίο διαβάζει τα αρχείο δεδομένων που υπάρχουν στο hdfs και τα χωρίζει σε τόσα μέρη όσοι και οι κόμβοι στη συστάδα. ο Local Hasher στη συνέχεια αντιγράφει έναν τεμαχισμό (partition) από το HDFS στο τοπικό σύστημα αρχείων κάθε κόμβου και στη συνέχεια το χωρίζει σε μικρότερου μεγέθους τμήματα-chunks με βάση τη ρύθμιση για το μεγαλύτερο μέγεθος chunk.

SMS (SQL to MapReduce to SQL) Planner: το Hadoop DB προσφέρει ένα εμπρόσθιο τμήμα(front end) παράλληλων βάσεων δεδομένων που επιτρέπει την εκτέλεση SQL επερωτήσεων. Ο SMS Planner επεκτείνει το Hive

Σε μετρήσεις που έγιναν συνέκριναν το HadoopDB με το Hadoop, το Vertica και το DBMSX εκτελώντας φόρτωση δεδομένων, εργασίες Grep, εργασίες επιλογής (selection task), εγασίες ένωσης (join task) και εργασίες συγκέντρωσης UDF (UDF aggregation task). Τα πειράματα έδειξαν ότι το HadoopDB μπορεί να πλησιάσει την απόδοση των συστημάτων παράλληλων βάσεων δεδομένων καθώς και να επιτύχει παρόμοια απόδοση με το Hadoop όσον αφορά την

αντοχή στα λάθη (fault tolerance) την ικανότητα να λειτουργεί σε ετερογενή περιβάλλοντα καθώς και το κόστος της άδειας λογισμικού.

3.2 HADOOP++

Το HadoopDB είχε σοβαρά μειονεκτήματα. Ανάγκαζε τους χρήστες να χρησιμοποιήσουν συστήματα παράλληλων βάσεων δεδομένων των οποίων η εγκατάσταση και η ρύθμιση δεν είναι εύκολα, άλλαζε τη διασύνδεση σε SQL καταργώντας έτσι ένα από τα μεγαλύτερα πλεονεκτήματα του Hadoop/MapReduce που είναι η απλότητα του προγραμματιστικού του μοντέλου , χρησιμοποιούσε τοπικά ACID-compliant DBMS engines παρότι μόνο indexing και join processing techniques ήταν απαραίτητες και τέλος απαιτούσε σημαντικές αλλαγές για να ταιριάξουν τα frameworks του Hadoop και του Hive. Ήταν ζητούμενη μια προσέγγιση όπου το σύστημα διατηρεί το interface του MapReduce/Hadoop, πλησιάζει την απόδοση των συστημάτων παράλληλων βάσεων και δεν τροποποιεί το υποκείμενο framework του Hadoop. Οι J.Dittrich et al. παρουσίασαν το Hadoop++ το οποίο βελτιώνει σημαντικά την απόδοση των εργασιών χωρίς να αλλάζει τίποτα στο framework του Hadoop [25]. Το επιτυγχάνει αλλάζοντας την εσωτερική δομή του split των δεδομένων και τροφοδοτώντας το Hadoop με τα κατάλληλα UDFs.

Το Hadoop++ προτείνει νέες τεχνικές ευρετηρίου (index) και ένωσης (join) : Trojan Index και Trojan Join αντίστοιχα για τη βελτίωση στο runtime των MR εργασιών. Για τη χρήση τους απαιτείται να οριστούν συναρτήσεις από τους χρήστες πέραν των map και reduce.

Trojan Index

Το Hadoop δεν προσφέρει τη δυνατότητα πρόσβασης ευρετηρίων λόγω της απουσίας σχήματος σε αντίθεση με τα ΣΔΒΔ. Το Hadoop++ υιοθετεί το Trojan Index για να ενσωματώσει τη δυνατότητα ευρετηρίου στο Hadoop. Για κάθε τμήμα δεδομένων δημιουργείται ένα Trojan Index ενώ προστίθεται και μια επικεφαλίδα η οποία περιέχει τα δεδομένα με το ευρετήριο, το μέγεθος του ευρετηρίου, το πρώτο και το τελευταίο κλειδί καθώς και τον αριθμό των εγγραφών. Τέλος προστίθεται και ένα footer που καθορίζει το όρια του split. Το Trojan Index είναι ένα ευρετήριο που αποτελείται από έναν διάσπαρτο κατάλογο πάνω από τα ταξινομημένα δεδομένα του split. Το ευρετήριο αποτυπώνεται με τη χρήση ενός CSS-tree όπου οι δείκτες των φύλλων του δείχνουν σε σελίδες μέσα στο split των δεδομένων. Έτσι κατά την εκτέλεση μιας επερώτησης

που αναφέρεται σε ένα ευρετηριασμένο σύνολο δεδομένων αρχικά αναγνωρίζονται τα όρια του split με τη χρήση του footer και δημιουργείται ένα map task για κάθε split.

Η προσέγγιση αυτή[25] :

- Δεν απαιτεί κάποια εξωτερική βιβλιοθήκη
- Δεν είναι διεισδυτική καθώς δεν αλλάζει το πλαίσιο του Hadoop αλλά υλοποιεί τη δομή του ευρετηρίου παρέχοντας το κατάλληλο UDFs
- Προσφέρει ένα προαιρετικό μονοπάτι πρόσβασης το οποίο μπορεί να χρησιμοποιηθεί επιλεκτικά από MR εργασίες
- Προσφέρει ενιαίο διαμερισμό καθώς το νέο λογικό τμήμα δεδομένων περιέχει το ευρετήριο και τα δεδομένα
- Δίνει τη δυνατότητα για μερικό ευρετήριο καθώς δεν είναι απαραίτητο να εφαρμοστεί ο Trojan Index σε ολόκληρο το τμήμα των δεδομένων.
- Δίνει τη δυνατότητα πολλαπλών ευρετηρίων στο ίδιο τμήμα δεδομένων ώστε να επιλέγεται το κατάλληλο ανά εργασία

Trojan join

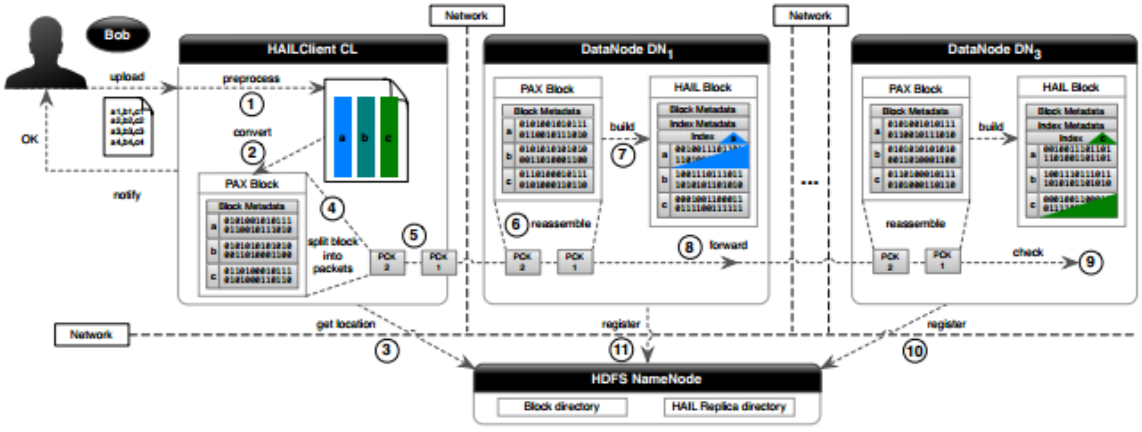
Το Trojan join είναι μια μέθοδος join που στηρίζεται στην υπόθεση ότι είναι γνωστά από πριν το σχήμα και ο φόρτος εργασίας. Βασική ιδέα είναι ο συν-τεμαχισμός των δεδομένων στο χρόνο φόρτωσης. Αν δοθούν δύο σχέσεις εισόδου εφαρμόζεται η ίδια συνάρτηση τεμαχισμού (partitioning) στα κοινά χαρακτηριστικά των σχέσεων στο χρόνο φόρτωσης των δεδομένων και τοποθετούνται τα ζεύγη που έχουν το ίδιο κλειδί ένωσης) από τις δύο σχέσεις στο ίδιο split συνεπώς και στον ίδιο κόμβο. Σαν αποτέλεσμα οι ενώσεις (joins) επεξεργάζονται τοπικά μέσα σε κάθε κόμβο στο χρόνο της επερώτησης ενώ επιπλέον έχουμε τη δυνατότητα να ομαδοποιήσουμε τα δεδομένα βάσει οποιουδήποτε χαρακτηριστικού στην ίδια MR εργασία. Η προσέγγιση αυτή[25]:

- Δεν είναι διεισδυτική
- Προσφέρει ενιαίο διαμερισμό των δεδομένων
- Δίνει στους χρήστες τη δυνατότητα να εκτελέσουν το join στον mapper
- Είναι συμβατή με το Trojan Index

Μετρήσεις που πραγματοποιήθηκαν και αφορούσαν φόρτωση δεδομένων, απόδοση σε εργασία επιλογής , απόδοση σε εργασίες ένωσης και ανοχή στα σφάλματα έδειξαν ότι το Hadoop++ έχει καλύτερη απόδοση από το Hadoop ενώ σε εργασίες ευρετηρίασης και ένωσης έχει καλύτερη απόδοση και από το HadoopDB. Επιπλέον αύξηση του μεγέθους του split βελτιώνει την απόδοση σε εργασίες ευρετηρίασης και επιλογής, μειώνει όμως την ανοχή σε σφάλματα.

3.3 HAIL (Hadoop Aggressive Indexing Library)

Το HAIL προτάθηκε από τους J.Dittrich et al και είναι μια ενίσχυση του HDFS και του Hadoop MapReduce [26]. Το Hail διατηρεί τα υπάρχοντα φυσικά αντίγραφα ενός HDFS Block σε διαφορετικές σειρές ταξινόμησης και με διαφορετικά συγκεντρωμένα ευρετήρια. Σαν αποτέλεσμα για τον προεπιλεγμένο παράγοντα αντιγραφής του 3 τουλάχιστον τρεις διαφορετικές σειρές ταξινόμησης και ευρετήρια είναι διαθέσιμα για την επεξεργασία της MapReduce εργασίας. Με τον τρόπο αυτό η πιθανότητα να βρεθεί κατάλληλο ευρετήριο αυξάνεται και ο χρόνος εκτέλεσης για συγκεκριμένο φόρτο εργασίας βελτιώνεται. Τροποποίησαν το HDFS upload pipeline ώστε να δημιουργεί τα ευρετήρια καθώς ανεβάζει αρχεία στο HDFS. Επομένως δεν απαιτείται επιπλέον διάβασμα των δεδομένων ούτε και επιπλέον MapReduce εργασίες για τη δημιουργία των ευρετηρίων.



Εικόνα 7: HAIL Upload pipeline [26]

Το HAIL έχει τα ακόλουθα πλεονεκτήματα [26]:

1. Συχνά βελτιώνει τους χρόνους τόσο για το ανέβασμα δεδομένων όσο και για τις επερωτήσεις. Το ανέβασμα δεδομένων είναι δραματικά ταχύτερο από το Hadoop++ και συχνά γρηγορότερο από το κανονικό Hadoop παρότι μετατρέπεται το αρχείο εισόδου σε δυαδικό PAX, δημιουργείται μια σειρά ακολουθιών ταξινόμησης και δημιουργούνται πολλαπλά ευρετήρια.
2. Δεν αλλάζει τις ιδιότητες αποτυχίας του Hadoop καθώς τα δεδομένα παραμένουν στο ίδιο λογικό HDFS Block και δεν αλλάζει η φυσική αντιπροσώπευση κάθε αντιγράφου του block επομένως για κάθε αντίγραφο μπορούμε να ανακτήσουμε το λογικό HDFS Block.
3. Δουλεύει με τις υπάρχουσες εργασίες MapReduce απαιτώντας μόνο ελάχιστες αλλαγές.

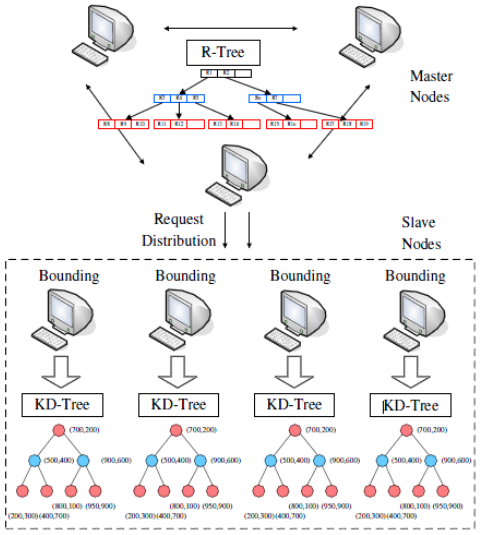
Συμπερασματικά το HAIL μειώνει τους απαιτούμενους χρόνους για το ανέβασμα δεδομένων και τη δημιουργία ευρετηρίων που ήταν σημαντικό μειονέκτημα στο Hadoop++. Το Hadoop++ δημιουργούσε ευρετήρια επιπέδου block αλλά απαιτούσε ακριβές MapReduce εργασίες για τη δημιουργία τους. Επιπλέον το Hadoop++ δημιουργούσε ευρετήρια ανά λογικό HDFS Block ενώ το HAIL δημιουργεί διαφορετικά ευρετήρια για κάθε φυσικό αντίγραφο. Πειραματικές συγκρίσεις απέδειξαν την υψηλή αποτελεσματικότητα του HAIL σε σχέση με το Hadoop και το Hadoop++ καθώς οι χρήστες μπορούν να ανεβάσουν δεδομένα και να τρέξουν εργασίες γρηγορότερα

3.4 EMINC και EEMINC

Οι X.Zhang et al παρουσίασαν μια προσέγγιση για δημιουργία πολυδιάστατου ευρετηρίου για υπολογιστικά συστήματα νέφους [27]. Χρησιμοποίησαν ένα συνδυασμό R-tree και KD-tree για την οργάνωση των εγγραφών δεδομένων που προσφέρει γρήγορη επεξεργασία επερωτήσεων και αποτελεσματική συντήρηση του ευρετηρίου. Η προσέγγιση αυτή μπορεί να επεξεργαστεί πολυδιάστατες επερωτήσεις όπως επερωτήσεις σημείου και επερωτήσεις εύρους.

Η δομή ευρετηρίου EMINC αποτελείται από ένα R-tree στους master κόμβους και ένα KD-tree στους slave κόμβους. Κάθε φύλλο του R-tree περιέχει έναν node-cube και έναν ή περισσότερους δείκτες που δείχνουν στους slave κόμβους που αντιστοιχούν στο node cube. Ο node cube είναι

ένα μια ακολουθία από μεσοδιαστήματα τιμών όπου κάθε μεσοδιάστημα αντιπροσωπεύει ένα φάσμα τιμών ενός χαρακτηριστικού αυτού του κόμβου.



Εικόνα 8: EMINC [27]

Με τις πληροφορίες του node cube στο EMINC η επεξεργασία επερωτήσεων μπορεί να βελτιωθεί σημαντικά με τον αποκλεισμό μη σχετικών κόμβων κατά τη φάση επιλογής κόμβων. Επίσης προκειμένου να διατηρηθεί η πληροφορία του κύβου και να παραμείνει χρήσιμη, η εισαγωγή και η διαγραφή δεδομένων στους slave κόμβους που πιθανώς τροποποιεί τον κύβο τους πρέπει να ενημερώνει τους master κόμβους για ενημέρωση του κύβου.

Για να αντιμετωπίσουν τους περιορισμούς του EMINC προτείνουν μια επέκταση του, το EEMINC [27]. Η διαφορά του από το EMINC είναι ότι στο EEMINC οι εγγραφές δεδομένων σε κάθε slave κόμβο απεικονίζονται από πολλαπλούς node cubes το μέγεθος και το σχήμα των οποίων εξαρτάται από τη μέθοδο που επιλέχθηκε για τον τεμαχισμό του αρχικού node cube. Η εκτέλεση επερωτήσεων στο EEMINC δε διαφέρει από αυτή στο EMINC, υπερτερεί όμως σε αποτελεσματικότητα καθώς η ύπαρξη node cubes υψηλότερης πιστότητας εξαλείφει σημαντικά την πιθανότητα να προωθηθούν οι επερωτήσεις σε κόμβους με μη σχετικό περιεχόμενο.

Μετρήσεις που πραγματοποίησαν εκτελώντας επερωτήσεις σημείου και επερωτήσεις εύρους επέδειξαν ότι η μέθοδος EMINC είναι αποτελεσματική για την εκτέλεση των συγκεκριμένων επερωτήσεων ενώ η EEMINC έχει ακόμα καλύτερη απόδοση και στις δύο περιπτώσεις.

3.5 B-TREE INDEXING

Οι S.Wu et al [5] παρουσιάζουν ένα σχήμα ευρετηρίασης βασισμένο σε B+-tree για αποτελεσματική επεξεργασία δεδομένων στο Cloud. Η προσέγγιση τους συνοψίζεται αρχικά στη δημιουργία ενός τοπικού B+-δέντρου για κάθε υπολογιστικό κόμβο το οποίο δείχνει μόνο σε δεδομένα που βρίσκονται στον κόμβο. Έπειτα οργανώνουν τους υπολογιστικούς κόμβους σε ένα δομημένο επίστρωμα και χτίζουν ένα Cloud Global index(CGI) για το σύστημα. Στο CGI δημοσιεύεται ένα τμήμα των τοπικών B+-δέντρων για αποτελεσματική επεξεργασία επερωτήσεων. Ανάλογα με το πρωτόκολλο δρομολόγησης του επιστρώματος το CG-index διαδίδεται στους υπολογιστικούς κόμβους. Τέλος προτείνουν και έναν προσαρμοστικό αλγόριθμο για επιλογή των δημοσιευμένων B+-tree κόμβων ανάλογα με την επερώτηση.
Πραγματοποίησαν μετρήσεις στην πλατφόρμα Amazon EC2 και συνέκριναν την προσέγγιση τους με το ScallableBTree το οποίο διατηρεί ένα μεγάλο B+-tree πάνω από το δίκτυο με βάση την απόδοση, την επεκτασιμότητα και την προσαρμοστικότητα και απέδειξαν ότι η προσέγγισή τους υπερισχύει.

3.6 RT-CAN

Οι J. Wang et al [28] προτείνουν το RT-CAN, ένα πολυδιάστατο σχήμα ευρετηρίασης στο epic το οποίο ενσωματώνει ένα CAN-based πρωτόκολλο δρομολόγησης και ένα σχήμα ευρετηρίασης βασισμένο στο R-tree για να υποστηρίξει αποτελεσματικά την επεξεργασία πολυδιάστατων επερωτήσεων σε ένα σύστημα νέφους. Κάθε υπολογιστικός κόμβος έχει μια δομή ευρετηρίου R-tree που ευρετηριάζει τα δεδομένα που είναι τοπικά αποθηκευμένα. Το RT-CAN προτείνει ένα μοντέλο το οποίο επιλέγει ωφέλιμους κόμβους για έκδοση και διατηρώντας ένα γενικό πολυδιάστατο ευρετήριο μπορεί να εντοπίζει τους υπολογιστικούς κόμβους που μπορεί να περιέχουν την απάντηση με μικρό κόστος.
Το ευρετήριο RT-CAN χτίζεται πάνω από τα τοπικά ευρετήρια R-trees. Με τον τρόπο αυτό η επεξεργασία μιας επερώτησης σπάει σε δύο μέρη. Στην πρώτη φάση ο επεξεργαστής ψάχνει στο γενικό ευρετήριο κάνοντας mapping την επερώτηση σε μερικούς CAN κόμβους. Αυτοί οι κόμβοι ψάχνουν το προσωρινά αποθηκευμένο R-tree τους και επιστρέφουν τις εγγραφές που ικανοποιούν την επερώτηση. Στη δεύτερη φάση με βάση τις εγγραφές του ευρετηρίου που ελήφθησαν η επερώτηση προωθείται στους αντίστοιχους κόμβους αποθήκευσης οι οποίοι

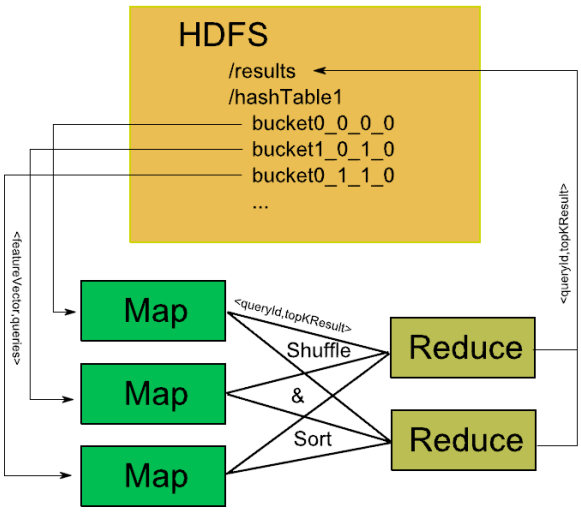
ανακτούν τα αποτελέσματα μέσω του τοπικού τους R-tree. Η εισαγωγή και η διαγραφή δεδομένων μπορεί να πυροδοτήσει μια ενημέρωση στο τοπικό R-tree που να παραβιάζει το γενικό ευρετήριο. Για να αποφευχθεί αυτό το ενδεχόμενο χρειάζεται να συγχρονίζεται το γενικό ευρετήριο με τα τοπικά R-trees. Σε περίπτωση που ένα R-tree έχει διαχωριστεί ή έχει συγχωνευθεί και είναι σημειωμένο ως εκδιδόμενο, εκδίδεται ξανά ώστε να αντικαταστήσει το ληγμένο γενικό ευρετήριο. Το κόστος της ανανέωσης του ευρετηρίου μειώνεται με την έκδοση μόνο ενός μέρους των κόμβων των R-trees[28].

Για την ελαχιστοποίηση του κόστους συντήρησης του RT-CAN ευρετηρίου προτείνουν μια προσαρμοστική στρατηγική ευρετηρίασης με βασική αρχή όταν επιλέγονται κόμβοι από το R-tree για το ευρετήριο πρέπει να είναι βέβαιο ότι το ευρετήριο που προκύπτει είναι πλήρες και μοναδικό. Η πληρότητα του ευρετηρίου επιτυγχάνεται όταν κάθε ακολουθία δεδομένων στην τοπική βάση του εξυπηρετητή περιέχεται σε ένα R-tree κόμβο του ευρετηρίου. Αντίστοιχα ένα ευρετήριο S είναι μοναδικό αν για κάθε R-tree κόμβο ni και τον πρόγονο του nj ισχύει: ni ∈ Sr →nj ∈ Sr ∧ nj ∈ Sr → ni ∈ Sr.

Πραγματοποίησαν μετρήσεις για να αποδείξουν τη δυνατότητα του RT-CAN ευρετηρίου να υποστηρίξει την εκτέλεση διαφόρων τύπων επερωτήσεων όπως σημείου, εύρους και K-κοντινότερων γειτόνων. Οι μετρήσεις πραγματοποιήθηκαν στην πλατφόρμα EC2 της Amazon και τα αποτελέσματα απέδειξαν την απόδοση και την αποτελεσματικότητα του RT-CAN ευρετηρίου.

3.7 RankReduce

Το πλαίσιο RankReduce είναι μια προσέγγιση που προτάθηκε από τους A.Stupar et al [29] για την αποτελεσματική επεξεργασία επερωτήσεων KNN(K-κοντινότερων γειτόνων). Η προσέγγιση αυτή προτείνει την εφαρμογή ενός ευρετηρίου LSH-based μέσα στο πλαίσιο του MapReduce. Ένα ευρετήριο βασισμένο σε LSH χρησιμοποιεί συναρτήσεις κατακερματισμού, ευαίσθητες ως προς την τοποθεσία για να ευρετηριάζει τα δεδομένα. Η εξέχουσα ιδιότητα των συναρτήσεων αυτών είναι ότι αντιστοιχούν με μεγάλη πιθανότητα παρόμοια αντικείμενα στον ίδιο κουβά κατακερματισμού. Η δημιουργία ευρετηρίου στην πραγματικότητα δημιουργεί αρκετά hash tables με χρήση διαφορετικών LSH συναρτήσεων προκειμένου να αυξηθεί η πιθανότητα σύγκρουσης για κοντινά σημεία.



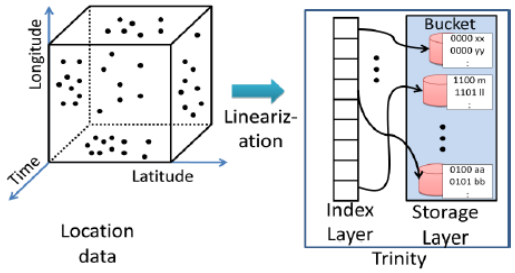
Εικόνα 9: Πλαίσιο RankReduce [29]

Κατά την επεξεργασία της επερώτησης η KNN αναζήτηση πραγματοποιείται με τον κατατεμαχισμό του σημείου της επερώτησης σε έναν κουβά ανά πίνακα και έπειτα με την ταξινόμηση όλως των αντικειμένων που ανακαλύφθηκαν με βάση την απόστασή τους από το σημείο της επερώτησης. Σαν αποτέλεσμα επιστρέφονται τα K κοντινότερα σημεία. Η εφαρμογή της εκτέλεσης της επερώτησης γίνεται σαν μια εργασία MapReduce. Πριν την έναρξη της MapReduce εργασίας υπολογίζονται οι τιμές κατατεμαχισμού για τα δεδομένα της επερώτησης. Αυτές οι τιμές χρησιμοποιούνται στη συνέχεια για την επιλογή από το LSH ευρετήριο των κουβάδων που είναι προς διερεύνηση. Οι κουβάδες που επιλέγονται περνιούνται σαν είσοδος στη MapReduce εργασία, παράγοντας πολλαπλά splits εισόδου. Τα splits αυτά διαβάζονται από μια ειδικά φτιαγμένη εφαρμογή της κλάσης InputFormat. Οι επερωτήσεις διαβάζονται μια φορά από την εφαρμογή της InputFormat και επαναχρησιμοποιούνται σα το τμήμα τη τιμής στην είσοδο της Map συνάρτησης ανάμεσα στις κλήσεις. Η είσοδος στη map συνάρτηση αποτελείται από το χαρακτηριστικό διάνυσμα που θα διερευνηθεί σαν κλειδί και τη λίστα των επερωτήσεων σαν τιμή. Η συνάρτηση map υπολογίζει την ομοιότητα του χαρακτηριστικού διανύσματος με τα διανύσματα των επερωτήσεων. Σταδιακά βγαίνουν οι K-κοντινότεροι γείτονες για κάθε διάνυσμα επερώτησης με τη μορφή από ζεύγη τιμής-αξίας. Η τελική ταξινόμηση στον reducer μπορεί να πραγματοποιηθεί και μέσα στο Hadoop αντί για τη μέθοδο Reduce σαν μια υποεργασία ταξινόμησης κλειδιών στον Reducer [29].

Πραγματοποίησαν μετρήσεις στις οποίες συνέκριναν την προσέγγισή τους με τη γραμμική σάρωση των δεδομένων μέσα σε μια MapReduce εργασία με χρήση συνθετικών και πραγματικών δεδομένων και απέδειξαν την καταλληλότητα της.

3.8 MD-HBase

Οι S.Nishimura et al [4] παρουσίασαν το MD-HBase για την υποστήριξη location based services (LBS). Τα παραδοσιακά DBMS με τα πολυδιάστατα ευρετήρια μπορούν να διαχειριστούν αποτελεσματικά τα χωρικά-χρονικά δεδομένα αλλά πιέζονται από τους υψηλούς ρυθμούς εισαγωγής, την απαίτηση για πραγματικού χρόνου επερωτήσεις καθώς και τον τεράστιο όγκο δεδομένων που τα συστήματα αυτά διαχειρίζονται. Στον αντίποδα τα key-value stores μπορούν να υποστηρίξουν αποτελεσματικά λειτουργία υψηλής κλίμακας αλλά δεν υποστηρίζουν προσβάσεις πολλαπλών χαρακτηριστικών που είναι απαραίτητες για την πλούσια λειτουργικότητα των LBS συστημάτων. Το MD-HBase είναι ένα επεκτάσιμο σύστημα διαχείρισης LBS συστημάτων που γεφυρώνει το κενό ανάμεσα σε κλίμακα και λειτουργικότητα. Χρησιμοποιεί ένα πολυδιάστατο ευρετήριο πάνω από ένα key-value store. Το υποκείμενο key-value store επιτρέπει στο σύστημα να διατηρεί υψηλό ρυθμό εισαγωγών και υποστηρίζει μεγάλο όγκο δεδομένων εξασφαλίζοντας παράλληλα ανοχή στα σφάλματα και υψηλή διαθεσιμότητα. Από την πλευρά του το επίπεδο του ευρετηρίου επιτρέπει αποτελεσματική επεξεργασία πολυδιάστατων επερωτήσεων. Το MD-HBase χρησιμοποίει σαν key-value store το HBase που είναι opensource ενώ στο επίπεδο του ευρετηρίου χτίζει δύο συνήθεις δομές ευρετηρίου, το K-d δέντρο και το Quad δέντρο. Επιπλέον το MD-HBase χρησιμοποίει τεχνικές «γραμμικοποίησης» όπως η Z-ordering για μετατροπή των πολυδιάστατων πληροφοριών τοποθεσίας σε ένα μονοδιάστατο διάστημα . Η μετατροπή των πολυδιάστατων σημείων των δεδομένων σε μια διάσταση είναι κομβική για το επίπεδο ευρετηρίου καθώς επιτρέπει τη χρήση μιας μονοδιάστατης βάσης δεδομένων για αποτελεσματική επεξεργασία πολυδιάστατων επερωτήσεων.



Εικόνα 10: MD-HBase αρχιτεκτονική [4]

Επίπεδο ευρετηρίου

Το επίπεδο του ευρετηρίου στο MD-HBase υποθέτει ότι το υποκείμενο επίπεδο αποθήκευσης δεδομένων αποθηκεύει τα αντικείμενα ταξινομημένα ως προς το κλειδί τους και διχοτομεί το διάστημα κλειδιών. Τα κλειδιά που αντιστοιχούν στη Z τιμή των διαστάσεων ευρετηριάζονται. Το ευρετήριο διχοτομεί το διάστημα σε εννοιολογικά υποδιαστήματα τα οποία με τη σειρά τους αντιστοιχίζονται σε ένα φυσικό χώρο αποθήκευσης, τον κουβά (bucket). Η αντιστοίχιση ενός εννοιολογικού διαστήματος στο επίπεδο του ευρετηρίου με έναν κουβά μπορεί να είναι ένα-προς-ένα, πολλά-προς-ένα ή πολλά-προς-πολλά ανάλογα με την υλοποίηση και τις απαιτήσεις της εφαρμογής. Ανέπτυξαν ένα σχήμα ονοματολογίας για να προσομοιώσουν ένα trie-based K-d δέντρο και ένα Quad δέντρο. Το σχήμα αυτό ονομάζεται longest common prefix naming. Εάν το πολυδιάστατο διάστημα χωρίζεται σε ίσου μεγέθους υποδιαστήματα και η κάθε διάσταση απαριθμείται με δυαδικές τιμές τότε η z-order κάθε υποδιαστήματος βρίσκεται ενθέτοντας τα bits των διαφορετικών διαστάσεων. Κάθε υποδιάστημα παίρνει το όνομα του από το μεγαλύτερο κοινό πρόθεμα των z τιμών των σημείων που περιέχονται στο υποδιάστημα.

Το MD-HBase εκμεταλλεύεται δύο βασικές ιδιότητες αυτού του σχήματος ονοματολογίας. Πρώτον αν το υποδιάστημα A περικλείει το υποδιάστημα B τότε το όνομα του A είναι πρόθεμα του ονόματος του B. Επομένως σε ένα split τα ονόματα των νέων υποδιαστημάτων προκύπτουν από το όνομα του αρχικού υποδιαστήματος επισυνάπτοντας τα bits των διαστάσεων στις οποίες έσπασαν. Δεύτερον το όνομα του υποδιαστήματος είναι αρκετό για να καθοριστούν τα όρια της περιοχής σε όλες τις διαστάσεις αυξάνοντας την απόδοση σε επερωτήσεις εύρους. Ο καθορισμός των ορίων αποτελείται από δύο βήματα: δοθέντος του ονόματος εξάγονται τα bits που αντιστοιχούν στις διαστάσεις και έπειτα επισυνάπτονται σε αυτά τα bits 0 για το κατώτερο όριο

και 1 για το ανώτερο. Το επίπεδο ευρετηρίου του MD-HBase εκμεταλλεύεται σχήμα ονοματολογίας για να αντιστοιχίσει δομές πολυδιάστατων ευρετηρίων σε ένα μονοδιάστατο υπόστρωμα.

Επίπεδο αποθήκευσης δεδομένων

Το επίπεδο αποθήκευσης δεδομένων του MD-HBase είναι ένα Key-value store τεμαχισμένο ανά φάσμα, το HBase, μια open source υλοποίηση του Bigtable. Ένας πίνακας στο HBase αποτελείται από μια συλλογή splits, τα επονομαζόμενα **regions**, όπου κάθε region αποθηκεύει ένα range partition του διαστήματος key- value. Η αρχιτεκτονική του απαρτίζεται από μια τριών επιπέδων δομή B+ δέντρου, τα regions που αποθηκεύουν δεδομένα αποτελούν το χαμηλότερο επίπεδο. Τα δύο υψηλότερα επίπεδα είναι ιδιαίτερα regions επονομαζόμενα ROOT και META. Μια εγκατάσταση HBase αποτελείται από ένα σύνολο εξυπηρετητών, τους **region servers**, που είναι υπεύθυνοι για την εξυπηρέτηση ενός υποσυνόλου regions. Τα regions αναθέτονται δυναμικά στους εξυπηρετητές, ο πίνακας META διατηρεί την αντιστοίχιση regions σε εξυπηρετητές. Σε περίπτωση που το μέγεθος ενός region ξεπεράσει ένα ορισμένο όριο το HBase το χωρίζει σε δύο υπό-περιοχές. Αυτό επιτρέπει στο σύστημα να μεγαλώνει δυναμικά με την είσοδο δεδομένων καθώς και να διαχειρίζεται την αλλοίωση των δεδομένων με τη δημιουργία καλύτερων τμημάτων (partititions) για δημοφιλή regions

Πραγματοποίησαν πειραματική ανάλυση της προσέγγισης τους σε συστάδα του Amazon EC2 εκτελώντας επερωτήσεις στο σύστημα τους τις οποίες εκτέλεσαν και σε MapReduce εργασίες στο Hadoop αποδεικνύοντας την βελτίωση στην απόδοση.

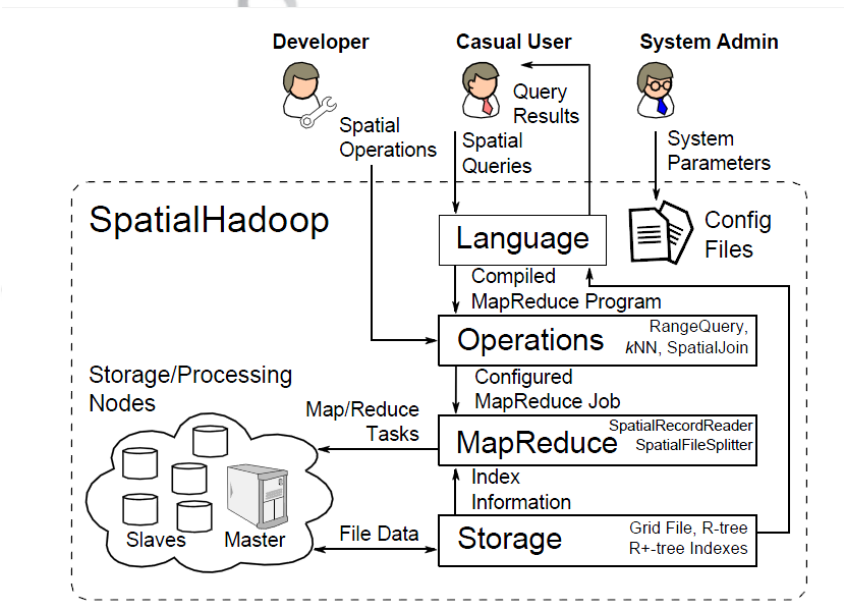
3.9 Spatial Hadoop

Το Spatial Hadoop είναι μια open source MapReduce προέκταση στο Hadoop για την επεξεργασία χωρικών δεδομένων [30]. Το SpatialHadoop τρέχει τα υπάρχοντα Hadoop προγράμματα ως έχουν αλλά επιτυγχάνει σημαντικά καλύτερη απόδοση στην επεξεργασία χωρικών δεδομένων. Χρησιμοποιεί μια απλή χωρική (spatial) γλώσσα υψηλού επιπέδου, μια δομή χωρικού ευρετηρίου δύο επιπέδων, βασικά χωρικά εξαρτήματα που υπάρχουν στο επίπεδο

του MapReduce και τρεις βασικές χωρικές λειτουργίες : επερωτήσεις εύρους, επερωτήσεις K-κοντινότερων γειτόνων και χωρικές ενώσεις.

Αρχιτεκτονική

Μια SpatialHadoop συστάδα περιλαμβάνει έναν master κόμβο που δέχεται την επερώτηση του χρήστη, την σπάει σε μικρότερες εργασίες και την προωθεί στους slave κόμβους [30]. Υιοθετεί ένα σχέδιο με στρώματα το οποίο αποτελείται από τέσσερα βασικά επίπεδα: γλώσσας (language), αποθήκευσης (storage), MapReduce και λειτουργιών (operations). Το επίπεδο της γλώσσας παρέχει μια απλή υψηλού επιπέδου γλώσσα που μοιάζει με SQL και υποστηρίζει χωρικούς τύπους δεδομένων και λειτουργίες. Το επίπεδο αποθήκευσης χρησιμοποιεί ένα ευρετήριο δύο επιπέδων από γενικά (global) και τοπικά χωρικά ευρετήρια. Το γενικό ευρετήριο μερίζει τα δεδομένα στους υπολογιστικούς κόμβους ενώ το τοπικό ευρετήριο οργανώνει τα δεδομένα μέσα στον ίδιο κόμβο. Το επίπεδο MapReduce έχει δύο νέα εξαρτήματα το SpatialFileSplitter και τον SpatialFileRecordReader που αξιοποιούν το γενικό και το τοπικό ευρετήριο αντίστοιχα για να αποκλείσουν τα δεδομένα που δε συνεισφέρουν στην απάντηση της επερώτησης. Το επίπεδο λειτουργιών εμπεριέχει την υλοποίηση διαφόρων χωρικών λειτουργιών (range query, *k*NN, spatial join) που εκμεταλλεύονται τα ευρετήρια και τα νέα εξαρτήματα του MapReduce επιπέδου.



Εικόνα 11: SpatialHadoop αρχιτεκτονική [30]

Επίπεδο γλώσσας: το SpatialHadoop παρέχει μια απλή υψηλού επιπέδου γλώσσα για να διευκολύνει την αλληλεπίδραση μη-τεχνικών χρηστών με το σύστημα. Η γλώσσα αυτή παρέχει υποστήριξη για χωρικούς τύπους δεδομένων , απλές χωρικές συναρτήσεις και λειτουργίες. Οι χωρικοί τύποι δεδομένων Point, Rectangle και Polygon χρησιμοποιούνται για να ορίσουν το σχήμα του αρχείου εισόδου κατά τη διαδικασία φόρτωσης. Οι συναρτήσεις Distance, Overlaps και MBR εφαρμόζονται στα χωρικά χαρακτηριστικά για να υπολογιστεί η απόσταση ανάμεσα στα κέντρα δύο σχημάτων, για να βρεθεί αν δύο σχήματα επικαλύπτονται και για να υπολογιστεί το μικρότερο οριακό παραλληλόγραμμο αντίστοιχα. Οι λειτουργίες range query, k-nearest neighbor και spatial join εφαρμόζονται σε αρχεία με χωρικά γνωρίσματα και παράγουν τα αποτελέσματα σε άλλο αρχείο εξόδου. Η γλώσσα του SpatialHadoop επεκτείνει την Pig Latin.

Επίπεδο αποθήκευσης: στο επίπεδο αποθήκευσης το SpatialHadoop προσθέτει νέα χωρικά ευρετήρια. Υπάρχουν δύο περιορισμοί που δεν επιτρέπουν στα παραδοσιακά ευρετήρια να χρησιμοποιηθούν ως έχουν στο Hadoop. Πρώτον τα παραδοσιακά ευρετήρια έχουν σχεδιαστεί για το διαδικαστικό προγραμματιστικό πρότυπο ενώ το SpatialHadoop ακολουθεί το MapReduce προγραμματιστικό πρότυπο. Δεύτερον τα παραδοσιακά ευρετήρια έχουν σχεδιαστεί για συστήματα τοπικών αρχείων ενώ το SpatialHadoop χρησιμοποιεί το HDFS το οποίο είναι περιορισμένο καθώς τα αρχεία μπορούν να γραφτούν με ένα συγκεκριμένο τρόπο επισύναψης και αφού γραφτούν δεν μπορούν να τροποποιηθούν περαιτέρω. Για να ξεπεράσει αυτούς τους περιορισμούς το SpatialHadoop οργανώνει το ευρετήριο του σε δύο επίπεδα, τοπικά και global. Το global ευρετήριο μερίζει τα δεδομένα σε κόμβους στη συστάδα, ενώ το τοπικό οργανώνει τα δεδομένα κάθε κόμβου. Το global ευρετήριο προετοιμάζει τη MapReduce εργασία ενώ τα τοπικά ευρετήρια χρησιμοποιούνται στις map εργασίες. Ο χωρισμός του αρχείου σε μικρότερα τμήματα επιτρέπει να ευρετηριάζεται κάθε τμήμα στη μνήμη και να γράφεται σε ένα αρχείο σειριακά. Το global ευρετήριο διατηρείται στην κύρια μνήμη του master κόμβου ενώ κάθε τοπικό ευρετήριο αποθηκεύεται σαν ένα blockfile σε έναν slave κόμβο. Το SpatialHadoop υποστηρίζει ευρετήρια grid file, R-tree και R+-tree. Ένα ευρετήριο παράγεται στο SpatialHadoop μέσα από μια MapReduce εργασία με τρεις φάσεις: partitioning, local indexing και global indexing. Στη φάση του partitioning κάθε αρχείο σπάει ώστε κάθε τμήμα του να χωράει σε ένα blockfile. Στη φάση του local indexing και ανάλογα με τον τύπο του ευρετηρίου που δημιουργείται ένα τοπικό ευρετήριο χτίζεται για κάθε partition και γράφεται σε ένα αρχείο

με ένα HDFS block. Στην τελική φάση του global indexing τα αρχεία που περιέχουν τα τοπικά ευρετήρια συνδέονται σε ένα μεγάλο αρχείο και ένα global ευρετήριο δημιουργείται ώστε να ευρετηριάζει κάθε partition χρησιμοποιώντας το MBR του σαν κλειδί.

Επίπεδο MapReduce: το SpatialHadoop προσθέτει δύο νέα εξαρτήματα στο επίπεδο του MapReduce, τον SpatialFileSplitter και SpatialRecordReader που εκμεταλλεύονται τα global και τα τοπικά ευρετήρια αντίστοιχα για αποτελεσματική πρόσβαση στα δεδομένα. Ο SpatialFileSplitter παίρνει σαν είσοδο ένα ή δύο χωρικά ευρετηριασμένα αρχεία μαζί με μια ορισμένη από το χρήστη συνάρτηση φιλτραρίσματος. Χρησιμοποιώντας το γενικό ευρετήριο αποκλείει τα blockfiles που δεν απαντούν στην επερώτηση. Ο SpatialRecordReader αξιοποιεί το τοπικό ευρετήριο επιτρέποντας την πρόσβαση στις εγγραφές σε ένα μπλοκ μέσω του ευρετηρίου αντί εξαντλητικά Ο SpatialFileSplitter και ο SpatialRecordReader μαζί βοηθούν τους προγραμματιστές να γράφουν πολλές χωρικές λειτουργίες σαν προγράμματα MapReduce.

Επίπεδο λειτουργιών: η χρήση ευρετηρίων στο επίπεδο αποθήκευσης και τα νέα εξαρτήματα δίνουν τη δυνατότητα για πολλές χωρικές λειτουργίες πάνω στο SpatialHadoop. Προσφέρονται οι λειτουργίες επερωτήσεις εύρους, k-κοντινότερων γειτόνων και χωρικές ενώσεις ενώ και άλλες χωρικές λειτουργίες όπως KNN ένωση και κοντινότερο μονοπάτι μπορούν να υλοποιηθούν ακολουθώντας παρόμοια προσέγγιση.

Συγκριτικές δοκιμές του SpatialHadoop με το Hadoop δείχνουν την υπεροχή του σε απόδοση όσον αφορά στην επεξεργασία χωρικών δεδομένων ενώ αποδεικνύουν ότι και στην περίπτωση επεξεργασίας μη χωρικών δεδομένων το SpatialHadoop δεν προσθέτει επιπλέον δαπάνη (overhead).

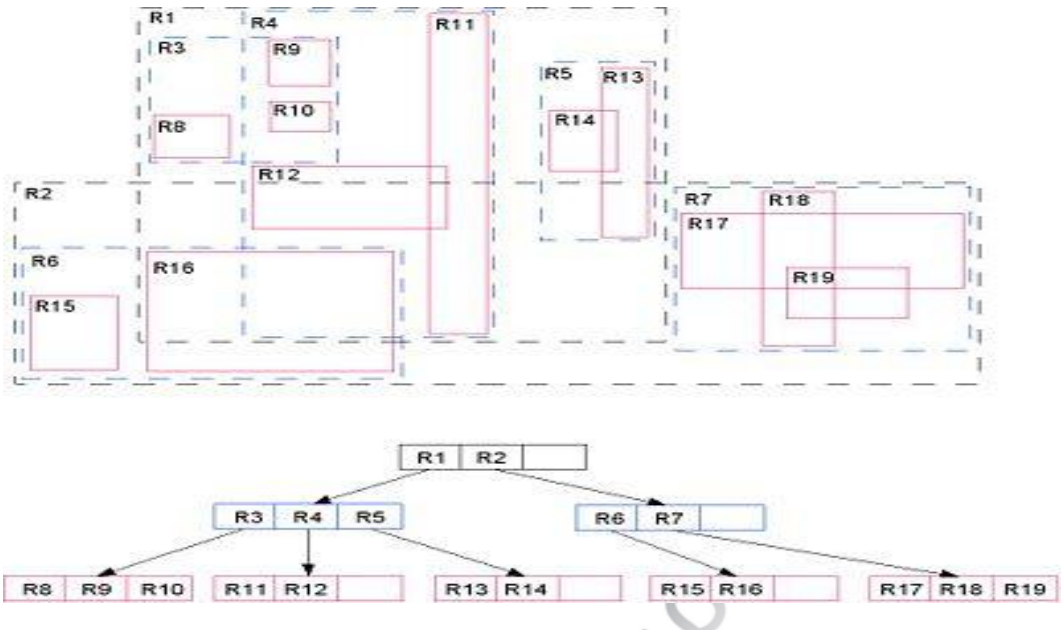
4. R-TREE ΚΑΙ ΧΩΡΙΚΕΣ ΕΠΕΡΩΤΗΣΕΙΣ

Το R-δέντρο είναι ένα καθ' ύψος ισοζυγισμένο δέντρο, παρόμοιο με το B-δέντρο με εγγραφές ευρετηρίου στους κόμβους φύλλα του που δείχνουν σε αντικείμενα δεδομένων [36]. Η δομή του είναι σχεδιασμένη με τέτοιο τρόπο ώστε η χωρική αναζήτηση να απαιτεί την επίσκεψη μόνο ενός μικρού αριθμού κόμβων. Το ευρετήριο είναι πλήρως δυναμικό και έτσι εισαγωγές και διαγραφές μπορούν να ανακατευτούν με αναζητήσεις χωρίς να απαιτείται περιοδική αναδιοργάνωση [34] [35].

Όπως και οι περισσότερες χωρικές μέθοδοι πρόσβασης το R-tree δεν ευρετηριάζει την ακριβή έκταση των αντικειμένων αλλά τα MBRs τους. Κάθε κόμβος αποτελείται από εγγραφές της μορφής (MBR, ptr). Στους κόμβους φύλλα το MBR είναι το μικρότερο δεσμευμένο ορθογώνιο παραλληλόγραμμο των αντικειμένων δεδομένων και ptr είναι το id του αντικειμένου. Σε κόμβους που δεν είναι φύλλα το MBR είναι το μικρότερο δεσμευμένο ορθογώνιο παραλληλόγραμμο όλων των αντικειμένων κάτω από τον R-tree κόμβο στον οποίο δείχνει το ptr.

Έστω M ο μέγιστος αριθμός εγγραφών που χωράν σε έναν κόμβο και $m < M/2$ η παράμετρος που καθορίζει τον ελάχιστο αριθμό εγγραφών σε έναν κόμβο τότε το R-δέντρο ικανοποιεί τα ακόλουθα [35][39]:

- Κάθε κόμβος-φύλλο περιέχει εγγραφές ανάμεσα σε m και M εκτός αν είναι η ρίζα
- Για κάθε εγγραφή του ευρετηρίου (l, tuple identifier) σε έναν κόμβο φύλλο, l είναι το μικρότερο παραλληλόγραμμο που χωρικά περιέχει τα n-διάστατα δεδομένα που αναπαριστά η ακολουθία
- Κάθε κόμβος που δεν είναι φύλλο έχει m έως M παιδιά εκτός αν είναι η ρίζα
- Για κάθε εγγραφή (l, child-pointer) σε έναν κόμβο που δεν είναι φύλλο το l είναι το μικρότερο παραλληλόγραμμο που χωρικά περιέχει τα παραλληλόγραμμα στους κόμβους-παιδιά
- Ο κόμβος ρίζα έχει τουλάχιστον δύο παιδιά εκτός αν είναι φύλλο
- Όλα τα φύλλα εμφανίζονται στο ίδιο επίπεδο



Εικόνα 12: R-tree δομή [34]

Το R-Δέντρο είναι η πιο διαδεδομένη μορφή χωρικού ευρετηρίου και αποτελεί την υποκείμενη μέθοδο ευρετηρίου πάνω από την οποία εκτελούνται οι γνωστότερες χωρικές επερωτήσεις. Δύο από τις σημαντικότερες μορφές τέτοιων επερωτήσεων είναι η επερώτηση εύρους και η επερώτηση κοντινότερου γείτονα. Η εκτέλεση των δύο αυτών επερωτήσεων πάνω από ένα παραδοσιακό R-Tree σαν δομή ευρετηρίου περιγράφεται ακόλουθα.

Επερώτηση εύρους

Μια επερώτηση εύρους αναζητά τα αντικείμενα σε μια σχέση R που ικανοποιούν μια χωρική απαίτηση λαμβάνοντας υπόψιν ένα ορισμένο αντικείμενο ή περιοχή [37] [31]. Αν η σχέση είναι ευρετηριασμένη από ένα R-tree μπορεί να χρησιμοποιηθεί το ευρετήριο ώστε να βρεθούν γρήγορα τα αντικείμενα που πληρούν τις προϋποθέσεις του φιλτραρίσματος. Το δέντρο διατρέχεται με ένα τρόπο «depth-first» ακολουθώντας τους δείκτες προς τις εγγραφές που διασταυρώνονται με το εύρος της επερώτησης[39]. Ο αναδρομικός αλγόριθμος αναζήτησης για το range query παίρνει τρεις παραμέτρους: το εύρος του query q, τη συνθήκη θ που τα αντικείμενα που ανακτώνται πρέπει να ικανοποιούν με το q και έναν κόμβο του R-tree n. Στην πρώτη κλήση της μεθόδου ο n κόμβος είναι η ρίζα του R-tree. Αν ο n είναι κόμβος φύλλο τότε για όλα τα MBRs των αντικειμένων που περιέχονται σε αυτόν και περνούν το βήμα του φιλτραρίσματος αποκτάται πρόσβαση στις ακριβείς γεωμετρίες των αντίστοιχων αντικειμένων

και τους εφαρμόζεται το βήμα του εξευγενισμού (refinement). Αν ο n δεν είναι κόμβος φύλλο τότε το MBR κάθε εγγραφής του αντιστοιχεί στο MBR όλων των αντικειμένων που ευρετηριάζονται στο υποδέντρο που δείχνει η εγγραφή. Μόνο αν το MBR αυτό περάσει στο φιλτράρισμα μπορεί να βρεθεί αντικείμενο στο αντίστοιχο υποδέντρο που να ικανοποιεί την επερώτηση. Στην περίπτωση αυτή η μέθοδος καλείται εξαντλητικά για τον κόμβο στον οποίο δείχνει η εγγραφή.

Για τον έλεγχο ενός MBR σε σχέση με την επερώτηση δε χρησιμοποιείται η συνθήκη θ αλλά η θ' η οποία καθορίζει μια συνθήκη που πρέπει να ικανοποιείται ανάμεσα στο query q και το MBR ενός αντικείμενου ώστε το αντικείμενο να ικανοποιεί την επερώτηση. Επιπλέον η θ' πρέπει να κρατάει ανάμεσα στο query και τον κόμβο n εάν αντικείμενο που ικανοποιεί την επερώτηση υπάρχει σε κόμβο φύλλο απόγονο του n.

Επερώτηση κοντινότερου γείτονα

Η επερώτηση για τον «κοντινότερο γείτονα» ανακτά από μία χωρική σχέση R το κοντινότερο αντικείμενο σε ένα query αντικείμενο q [36]. Επίσημα ο κοντινότερος γείτονας του q στην R ορίζεται από τον τύπο

{o ∈ R : ∀ o' ∈ R, dist (o, q) ≤ dist (o', q)}, όπου dist (o, q) είναι η Ευκλείδεια απόσταση ανάμεσα στο o και στο q[39]. Ο ορισμός μπορεί να επεκταθεί και να συμπεριλάβει και μια παράμετρο k η οποία θα ορίζει τον αριθμό των κοντινότερων γειτόνων που θα ανακτηθούν.

Σε περίπτωση που η χωρική σχέση δεν είναι ευρετηριασμένη πρέπει να περάσουμε από όλα τα αντικείμενα σε αυτή ώστε να βρούμε τον κοντινότερο γείτονα στο query αντικείμενο q ενώ δε συμβαίνει το ίδιο αν η σχέση είναι ευρετηριασμένη με ένα R-tree.Υπάρχει μια ιδιότητα στους κόμβους του R-tree που βοηθά στην επιτάχυνση της αναζήτησης. Έστω q το query αντικείμενο τότε για κάθε κόμβο n του R-tree ισχύει

$$dist(q, MBR(n)) \leq dist(q, o_i), \forall o_i \text{ indexed under } n$$

Επομένως κοιτώντας το MBR ενός R-tree κόμβου n μπορούμε να καταλήξουμε για την ελάχιστη πιθανή απόσταση ανάμεσα στο q και σε κάθε αντικείμενο που είναι ευρετηριασμένο κάτω από τον n. Βασικό πλεονέκτημα είναι ότι αν η απόσταση είναι μεγαλύτερη από τις γνωστές

αποστάσεις για τα αντικείμενα που έχουμε ήδη δει μπορούμε να αποφύγουμε να ελέγξουμε τα αντικείμενα που ευρετηριάζονται κάτω από αυτόν τον κόμβο.

Ο "depth-first" αλγόριθμος αναζήτησης (DF) επερώτησης κοντινότερου γείτονα λειτουργεί περίπου σαν τον αλγόριθμο της επερώτησης εύρους με τη διαφορά ότι χρησιμοποιεί πληροφορίες για τον μέχρι τώρα κοντινότερο γείτονα για να περιορίσει την αναζήτηση και να αποκλείσει τμήματα του R-Tree. Αρχικά ο κοντινότερος γείτονας είναι ένα φανταστικό αντικείμενο με άπειρη απόσταση από το q και ο αλγόριθμος καλείται παίρνοντας σαν παράμετρο n τη ρίζα του δέντρου. Εάν ο τρέχων κόμβος είναι φύλλο τότε οι εγγραφές του (πρώτα τα MBRs του και ύστερα τα πραγματικά αντικείμενα) ελέγχονται αν βρίσκονται πιο κοντά στο q από τον μέχρι στιγμής κοντινότερό του γείτονα. Εάν βρίσκονται ανανεώνονται οι πληροφορίες του NN του q. Εάν ο τρέχων κόμβος δεν είναι φύλλο τότε μόνο οι εγγραφές των οποίων τα MBRs είναι κοντύτερα στο q από τον μέχρι στιγμής NN μπορεί να δείχνουν σε υποδέντρο που να περιέχει τον NN.

Ο DF αλγόριθμος αναζήτησης είναι αποτελεσματικός αλλά αποδεικνύεται ότι δεν περνάει από τον μικρότερο δυνατό αριθμό κόμβων του R-tree σε κάθε περίσταση[39]. Ένας branch-and-bound αλγόριθμος που λειτουργεί με τρόπο ανάμεσα σε depth-first και breadth-first αναζήτησης περνά από τους R-tree κόμβους με την πιο υποσχόμενη σειρά για να βρεθεί ο κοντινότερος γείτονας του query αντικειμένου q. Βασική ιδέα η διατήρηση μιας στοίβας (π.χ ουράς προτεραιότητας) Q που οργανώνει δυναμικά τις εγγραφές και τα αντικείμενα των επισκεπτόμενων R-tree κόμβων λαμβάνοντας υπόψιν την απόστασή τους από το q. Αρχικά η στοίβα Q περιέχει όλες τις εγγραφές της ρίζας του R-tree, το αντικείμενο στην κορυφή είναι η εγγραφή της ρίζας της οποίας το MBR βρίσκεται πιο κοντά στο q. Όσο υπάρχουν ακόμα στοιχεία στη στοίβα Q επισκέπτεται το στοιχείο στην κορυφή και το αφαιρεί από τη στοίβα. Αν το στοιχείο αυτό δεν είναι φύλλο τότε επισκέπτεται τον κόμβο στον οποίο δείχνει και τα περιεχόμενα του προστίθενται στα στοίβα. Εάν είναι κόμβος φύλλο τότε το αντικείμενο στο οποίο δείχνει ελέγχεται, υπολογίζεται η ακριβής του απόσταση από το q και προστίθεται στη στοίβα. Τελικά αν το στοιχείο στην κορυφή της στοίβας είναι αντικείμενο αποτελεί εγγυημένα τον κοντινότερο γείτονα του q και ο αλγόριθμος τερματίζεται. Ο λόγος είναι ότι κάθε αντικείμενο, MBR αντικειμένου ή MBR κόμβου που περιέχεται στη στοίβα έχει μεγαλύτερη απόσταση από το q επομένως δεν είναι πιθανό να υπάρχει άλλο αντικείμενο κοντύτερα στο q.

5.ΠΑΡΟΥΣΙΑΣΗ ΠΡΟΒΛΗΜΑΤΟΣ

Η παρούσα εργασία πραγματεύεται την αξιοποίηση του υπολογιστικού νέφους και πλαισίων όπως το MapReduce για την διαχείριση πολυδιάστατων δεδομένων και την εκτέλεση επερωτήσεων σε αυτά. Πολλά από τα δεδομένα που οι εταιρίες καλούνται να επεξεργαστούν είναι πολυδιάστατα και πολύπλοκα και πλέον παράγονται με ταχύτατους ρυθμούς με αποτέλεσμα η επεξεργασία των επερωτήσεων να αντιμετωπίζει πρόβλημα με τον μεγάλο όγκο τους. Για παράδειγμα τα Location-Based Services χρησιμοποιούν πολυδιάστατα δεδομένα και συναντώνται σε πολλές πραγματικές εφαρμογές όπως είναι το Facebook, το Flickr, το Twitter κ.α [4].

Το υπολογιστικό νέφος μπορεί να διευκολύνει την εκτέλεση των επερωτήσεων αυτών προσφέροντας μεγάλη υπολογιστική δύναμη και δυνατότητα αποθήκευσης. Επιπλέον πλαίσια όπως το MapReduce που υποστηρίζουν παράλληλη επεξεργασία ενδείκνυνται για την επεξεργασία μεγάλου όγκου δεδομένων [31].

Σημαντικό όμως μειονέκτημα του MapReduce και του Hadoop είναι η απουσία υποστήριξης επιλεκτικής πρόσβασης στα δεδομένα [6] [7]. Η απουσία ευρετηρίων είναι ένα βασικό εμπόδιο που μειώνει την αποδοτικότητά τους κατά την εκτέλεση επερωτήσεων σε σχέση με τα ΣΔΒΔ ιδιαίτερα όταν οι επερωτήσεις είναι επιλεκτικές και απαιτούν πρόσβαση σε ένα υποσύνολο μόνο των δεδομένων. Αντίθετα οι MapReduce εργασίες επεξεργάζονται ολόκληρο το σύνολο των δεδομένων για να φτάσουν στο αποτέλεσμα.

Το πρόβλημα στο οποίο καλείται να απαντήσει η παρούσα εργασία είναι το κατά πόσο θα μπορούσε το Hadoop να συνδυαστεί με ένα ευρετήριο τύπου R-tree ώστε να μπορέσει να αξιοποιηθεί η υπολογιστική ισχύς που η παράλληλη επεξεργασία σε συστάδα προσφέρει με την υψηλή αποδοτικότητα σε εκτέλεση επερωτήσεων που προσφέρει η χρήση ευρετηρίων.

Επιλέχθηκε το R-tree σα δομή ευρετηρίου καθώς αποτελεί την πιο διαδεδομένη μορφή χωρικού ευρετηρίου και η επερώτηση εύρους, μια από τις συνηθέστερες μορφές επερώτησης σε χωρικά δεδομένα. Υπάρχει ήδη κώδικας που εκτελεί τη συγκεκριμένη επερώτηση με χρήση R-tree ευρετηρίου τοπικά και στόχος της παρούσας μελέτης είναι ο κώδικας αυτός να μπορέσει να τρέξει παράλληλα σε συστάδα με τη μορφή μιας MapReduce εργασίας παράγοντας τα ίδια

αποτελέσματα που παράγει τοπικά για δεδομένο αρχείο εισόδου. Επιπλέον πρέπει να μετατραπεί και σε MR Job και ο κώδικας που εκτελεί την επερώτηση εύρους χωρίς χρήση ευρετηρίου προκειμένου να γίνει σύγκριση των δύο μεθόδων και να βρεθεί αν η χρήση ευρετηρίου όντως βελτιώνει την αποδοτικότητα.

Η ελαχιστοποίηση του χρόνου εκτέλεσης της επερώτησης θα επιτευχθεί μέσω της ελαχιστοποίησης της εισόδου της συνάρτησης map καθώς και των bytes που διαβάζονται από το HDFS ώστε να εξασφαλιστεί ότι η πρόσβαση θα περιοριστεί στο τμήμα των δεδομένων που είναι απαραίτητο και όχι στο σύνολο τους.

Για την αξιολόγηση θα πραγματοποιηθεί πειραματική μελέτη κατά την οποία θα εκτελούνται οι δύο MapReduce εργασίες που επεξεργάζονται την επερώτηση εύρους με χρήση ευρετηρίου και χωρίς σε διαφορετικές κάθε φορά συνθήκες. Συγκεκριμένα θα εκτελούνται με διαφορετικό αρχείο εισόδου στο οποίο θα διαφέρει ο αριθμός των σημείων και οι διαστάσεις τους. Στόχος της πειραματικής μελέτης είναι να αποδείξει ότι η χρήση του R-tree ευρετηρίου βελτιώνει την απόδοση κατά την εκτέλεση της επερώτησης το οποίο συνεπάγεται μικρότερο χρόνο εκτέλεσης, μικρότερο αριθμό bytes για το map input και μικρότερο αριθμό bytes που διαβάζονται από το HDFS.

Οι στόχοι της εργασίας συνοψίζονται ακόλουθα

- **Βελτίωση της απόδοσης του Hadoop στην εκτέλεση επερωτήσεων σε πολυδιάστατα δεδομένα**
- **Αξιολόγηση της χρήσης ευρετηρίου για εκτέλεση επερωτήσεων σε πολυδιάστατα δεδομένα με χρήση του Hadoop**

Για να επιτευχθούν οι προαναφερθέντες στόχοι πρέπει να γίνουν τα εξής:

- Προσαρμογή του κώδικα για εκτέλεση επερώτησης εύρους χωρίς τη χρήση ευρετηρίου σαν MapReduce εργασία
- Προσαρμογή του κώδικα για εκτέλεση επερώτησης εύρους με τη χρήση ευρετηρίου σαν MapReduce εργασία

Η βελτίωση της απόδοσης του Hadoop κατά την εκτέλεση των επερωτήσεων θα επιτευχθεί με τους εξής τρόπους:

- Ελαχιστοποίηση του χρόνου εκτέλεσης
- Μείωση των δεδομένων εισόδου στη συνάρτηση map
- Μείωση των δεδομένων που διαβάζονται από το HDFS κατά την εκτέλεση της επερώτησης

Βασικότερη πρόκληση στην προσέγγιση μας είναι αν το Hadoop μπορεί να εκμεταλλευτεί τα πλεονεκτήματα των πολυδιάστατων ευρετηρίων για να βελτιώσει την απόδοση του στην εκτέλεση επερωτήσεων καθώς και να μη χρειάζεται να διατρέχει όλα τα δεδομένα κατά την εκτέλεση τους.

6.ΣΧΕΔΙΑΣΗ

Για την αξιοποίηση της χρήσης του R-tree ευρετηρίου για επερωτήσεις σε πολυδιάστατα δεδομένα πάνω στο υπολογιστικό νέφος ακολουθήθηκε σχεδίαση σε επίπεδο κώδικα η οποία θα παρουσιαστεί αναλυτικά στο παρόν κεφάλαιο.

Ο κώδικας είναι χωρισμένος σε δύο μέρη. Το πρώτο μέρος αναλαμβάνει τη δημιουργία του ευρετηρίου και το ανέβασμα των απαραίτητων αρχείων στο HDFS ενώ το δεύτερο την εκτέλεση των χωρικών επερωτήσεων με τη μορφή μιας MapReduce εργασίας. Αντίστοιχα μια MapReduce εργασία σχεδιάστηκε και για τη σειριακή αναζήτηση σημείων που ικανοποιούν τη χωρική συνθήκη χωρίς τη χρήση ευρετηρίου.

Στην προσέγγισή μας προτείνεται να μην ανεβαίνει στο HDFS το ίδιο το προς επεξεργασία αρχείο πολυδιάστατων δεδομένων αλλά το ευρετήριο του βασιζόμενο στο R-tree με τη βοήθεια του οποίου θα εκτελούνται και οι χωρικές επερωτήσεις. Η παραγωγή του αρχικού αρχείου προς επεξεργασία γίνεται από γεννήτριες τυχαίων αριθμών στις οποίες ο χρήστης ορίζει τον αριθμό των διαστάσεων των σημείων, τον αριθμό των σημείων, την ανώτατη τιμή που μπορεί να πάρουν οι διαστάσεις καθώς και την κατανομή των σημείων αν το επιθυμεί.

Τα αρχεία αυτά δημιουργούνται τοπικά σε ένα ορισμένο μονοπάτι από το χρήστη. Το μονοπάτι αυτό χρησιμοποιείται στην πρώτη φάση της σχεδίασης για τη δημιουργία ευρετηρίων και για το ανέβασμα τους στο HDFS. Για κάθε αρχείο σε αυτό το μονοπάτι στο δίσκο δημιουργείται ένα R-tree based ευρετήριο. Κατά τη δημιουργία του R-tree ευρετηρίου του αρχείου είναι σημαντικό το ευρετήριο που θα προκύψει να μην περνάει σε μέγεθος το blocksize του HDFS. Για το λόγο αυτό ελέγχουμε το μέγεθος του αρχικού αρχείου και αν αυτό ξεπερνά ένα πειραματικά ορισμένο κλάσμα του HDFS blocksize το σπάμε σε ανάλογου μεγέθους splits. Για τα νέα υπο-αρχεία που δημιουργούνται χτίζουμε τα ευρετήριά τους. Το ευρετήριο που δημιουργείται για κάθε αρχείο αποτελείται από 8 αρχεία (με καταλήξεις .meta,.ctr,.edt,.flt,.mtd,.rbm,.rdt,.ubm.)

Τα αρχεία αυτά του ευρετηρίου στη συνέχεια ανεβαίνουν στο HDFS σε διαφορετικά όμως μονοπάτια. Το αρχείο με τα μεταδεδομένα κάθε ευρετηρίου με κατάληξη .meta τοποθετείται στο InputPath της MapReduce εργασίας καθώς περιέχει τα MBRs και είναι απαραίτητο για να ξεκινήσει η εκτέλεση της επερώτησης, ενώ τα υπόλοιπα σε ένα φάκελο Index ο οποίος δε

συμπεριλαμβάνεται στο InputPath της MR εργασίας. Εδώ τελειώνει το πρώτο μέρος της σχεδίασης και ακολουθεί η φάση της εκτέλεσης της επερώτησης.

Η εκτέλεση της επερώτησης γίνεται ακολουθώντας τη λογική των συναρτήσεων MapReduce και υλοποιώντας ένα ειδικά σχεδιασμένο InputFormat. Το InputFormat είναι εκείνο που τροφοδοτεί τη συνάρτηση map με ζεύγη τιμής κλειδιού. Το InputFormat που σχεδιάστηκε τρέχει στα αρχεία μεταδεδομένων που βρίσκονται στο InputPath της MR εργασίας. Ανοίγει κάθε ένα από αυτά τα αρχεία και δεν τα επεξεργάζεται γραμμή γραμμή παρά ανακτά από αυτά τα μεταδεδομένα που είναι απαραίτητα για την φόρτωση του R-tree και την εκτέλεση της επερώτησης. Έχοντας ανακτήσει τα απαραίτητα μεταδεδομένα εκτελεί την επερώτηση και συγκεντρώνει τα σημεία που ικανοποιούν τη συνθήκη της. Κάθε ένα από αυτά τα σημεία προστίθεται στην τιμή που θα σταλεί στη συνάρτηση map ενώ ορίζεται και σαν κλειδί ο αριθμός 1.

Κάθε συνάρτηση map παίρνει σαν είσοδο ένα ζεύγος τιμής - κλειδιού όπου σαν τιμή έχει οριστεί το σύνολο των σημείων που απαντούν στην επερώτηση σε μορφή κειμένου όπου κάθε σημείο αναπαριστά μια γραμμή. Ανακτά ένα ένα αυτά τα σημεία και για κάθε ένα από αυτά εν συνεχεία περνάει τα ζεύγη τιμής – κλειδιού στη συνάρτηση Reduce. Στην περίπτωση της επερώτησης εύρους το κλειδί είναι κοινό για όλα τα σημεία και η συνάρτηση Reduce δεν επεξεργάζεται επιπλέον τα αποτελέσματα. Σε περιπτώσεις άλλων επερωτήσεων όπως για παράδειγμα K- κοντινότερων γειτόνων το κλειδί μπορεί να χρησιμοποιηθεί για την ταξινόμηση ων σημείων με βάση την απόσταση.

Αντίστοιχα σχεδιάστηκε και μια δεύτερη MapReduce εργασία για τη μέθοδο ης σειριακής αναζήτησης για την επερώτηση. Σε αυτή την περίπτωση ο χρήστης πρέπει να έχει ανεβάσει ολόκληρο το αρχείο των σημείων στο HDFS και συγκεκριμένα στο Input Path της MapReduce εργασίας. Στη συνέχεια η συνάρτηση map( ) διατρέχει το αρχείο γραμμή - γραμμή και βρίσκει το σημείο που αναπαριστά κάθε γραμμή. Υπολογίζει αν το σημείο αυτό ικανοποιεί τη συνθήκη της επερώτησης και σε περίπτωση που την ικανοποιεί του αναθέτει ένα κλειδί (στην προκειμένη περίπτωση κοινό για όλα τα σημεία και ίσο με 1) και στέλνει τα ζεύγη κλειδιού-τιμής που προκύπτουν σαν είσοδο στη συνάρτηση Reduce η οποία απλά συγκεντρώνει τα ζεύγη κλειδιού – τιμής χωρίς να κάνει κάποια επιπλέον ταξινόμηση.

6.1 ΑΡΧΙΚΟΠΟΙΗΣΗ

Στην αρχή του κώδικα αρχικοποιούνται μεταβλητές που χρησιμοποιούνται κατά την εκτέλεση του και πραγματοποιείται και η διαμόρφωση (configuration) των MapReduce εργασιών. Καθορίζεται η διάσταση των σημείων του αρχείου εισόδου, ο αριθμός τους και η ανώτατη τιμή τους. Ορίζεται επίσης το τοπικό μονοπάτι στο οποίο βρίσκονται τα αρχεία εισόδου καθώς και το μονοπάτι στο οποίο τοποθετούνται τα splits του αρχείου σε περίπτωση που είναι μεγαλύτερο από το block του HDFS και χρειάζεται να διασπαστεί σε μικρότερα τμήματα. Επιπλέον ορίζεται το τοπικό μονοπάτι στο οποίο θα δημιουργηθούν τα ευρετήρια για κάθε υποαρχείο αλλά και η τοποθεσία στο HDFS στην οποία θα ανέβουν τα αρχεία αυτά. Η τοποθεσία αυτή διαφέρει ανάλογα με τον τύπο του αρχείου καθώς το αρχείο με τα μεταδεδομένα του ευρετηρίου τοποθετείται στο InputPath της MapReduce εργασίας με τη χρήση ευρετηρίου ενώ τα υπόλοιπα αρχεία ανεβαίνουν σε διαφορετικό φάκελο στο HDFS. Επιπλέον γίνεται η διαμόρφωση για τις δύο MapReduce εργασίες που θα τρέξουν για την εκτέλεση των επερωτήσεων με χρήση και χωρίς χρήση R-tree ευρετηρίου με τη βοήθεια δύο instances της κλάσης του Hadoop JobConf. Για κάθε μία από τις εργασίες ορίζεται το InputPath, το OutputPath που θα πρέπει να έχει δημιουργηθεί ήδη στο HDFS, οι κλάσεις που θα λειτουργήσουν ως Mapper, Reducer και Partitioner καθώς και οι τύποι των τελικών key-value.

6.2 ΔΗΜΙΟΥΡΓΙΑ ΔΕΔΟΜΕΝΩΝ ΕΙΣΟΔΟΥ

Τα δεδομένα που επεξεργαζόμαστε στην παρούσα εργασία είναι πολυδιάστατα και παράγονται τυχαία από data generators. Κατά τη δημιουργία του αρχείου ο χρήστης επιλέγει τον αριθμό των σημείων, τον αριθμό των διαστάσεών τους καθώς και τη μέγιστη τιμή τους. Επιπλέον καθορίζει και τον τύπο της κατανομή των σημείων (distribution) επιλέγοντας ανάμεσα σε uniform, correlated και anticorrelated κατανομή. Για τις ανάγκες της πειραματικής μελέτης στην παρούσα εργασία δημιουργήθηκαν αρχεία με διαστάσεις σημείων 4 ή 6 και διαφορετικό αριθμό σημείων κάθε φορά. Κάθε γραμμή του αρχείου αναπαριστά ένα σημείο και αποτελείται από το id και τις συντεταγμένες του, ανάλογα με τη διάσταση των σημείων που επιλέχθηκε. Ακόλουθα παρουσιάζεται ένα μέρος του αρχείο με τα σημεία με διάσταση σημείων ίση με 4.

10 123.43107720925241 809.8667320325083 516.2391235153824 907.5147767091454

11 220.97864831279975 144.54732020245442 652.2861512701562 37.58419455979351

12 253.88135968302646 143.7909050184992 258.1199806368988 855.5372257487121

13 494.38653340217013 906.4550085511513 806.4240674544607 245.14251179409615

14 912.1827395858313 163.62108286290155 698.3913954912113 718.3996480553828

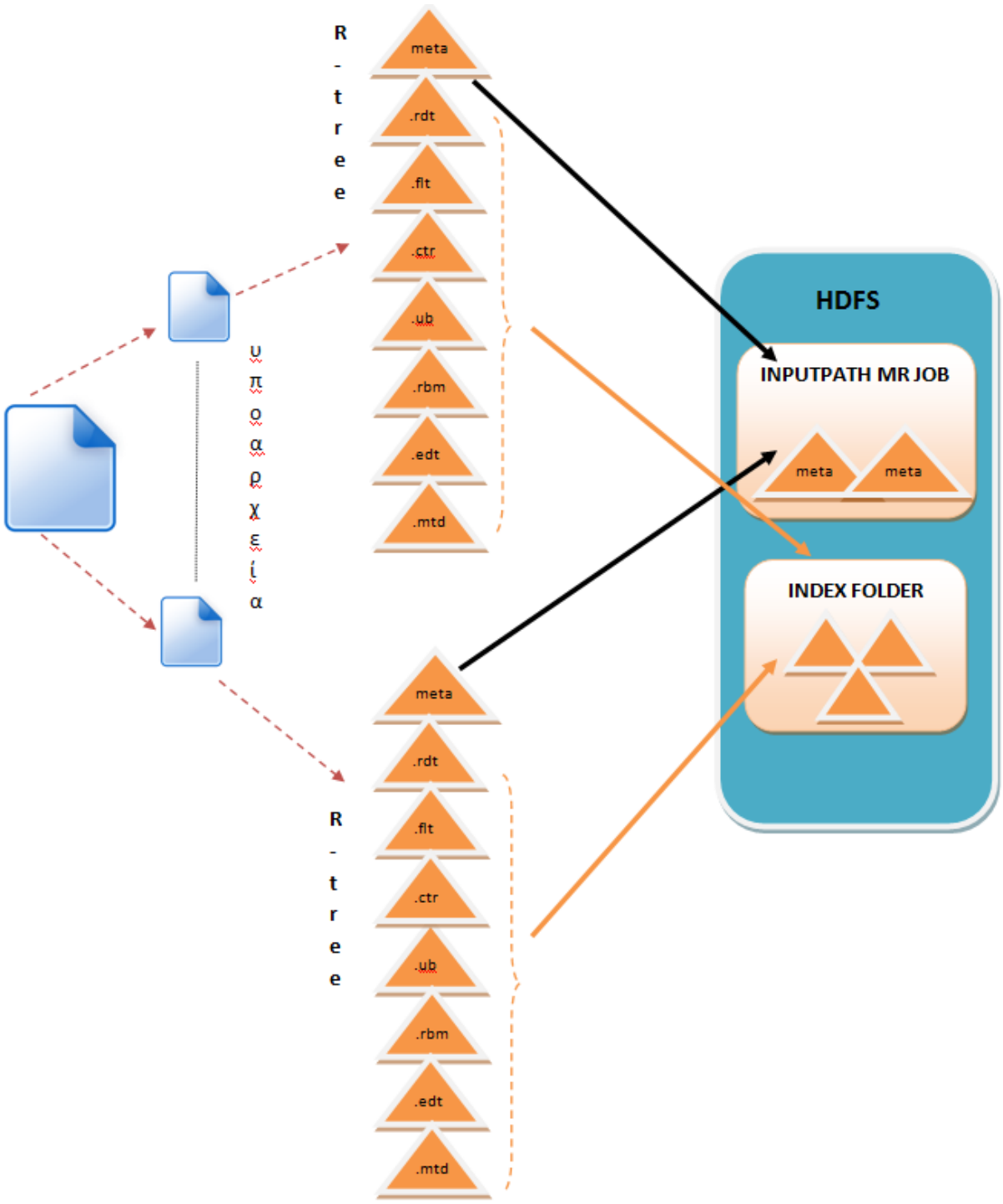
15 436.6328760314052 841.1746838377892 896.415940662804 262.05088760510444

6.3 ΤΕΜΑΧΙΣΜΟΣ ΑΡΧΕΙΩΝ

Για κάθε αρχείο που βρίσκεται στο μονοπάτι των αρχείων εισόδου όπως αυτό έχει οριστεί κατά το configuration γίνεται έλεγχος του μεγέθους του. Σε περίπτωση που το μέγεθος του υπερβαίνει το μέγεθος του blocksize το αρχείο σπάει σε μικρότερα υποαρχεία συγκεκριμένου μεγέθους τα οποία τοποθετούνται τοπικά στο δίσκο, σε μονοπάτι ορισμένο από το χρήστη. Η διαδικασία αυτή εκτελείται ώστε το ευρετήριο που θα παραχθεί για κάθε αρχείο να χωράει σε ένα block του HDFS και να μη καταμερίζεται σε περισσότερα.

6.4 ΔΗΜΙΟΥΡΓΙΑ R-TREE ΕΥΡΕΤΗΡΙΟΥ

Για κάθε αρχείο που βρίσκεται στο τοπικό μονοπάτι που τοποθετήθηκαν τα υποαρχεία που προέκυψαν από τον διαχωρισμό των αρχικών αρχείων εισόδου χτίζεται ένα R-tree based ευρετήριο. Κάθε ευρετήριο αποτελείται από 8 διακριτά αρχεία διαφορετικού μεγέθους με καταλήξεις .meta,.ctr,.edt,.flt,.mtd,.rbm,.rdt,.ubm. Το συνολικό μέγεθος του ευρετηρίου υπερβαίνει το μέγεθος του αρχικού αρχείου, είναι όμως πάντα μικρότερο από το μέγεθος του HDFS Blocksize. Στη συνέχεια τα ευρετήρια που δημιουργήθηκαν ανεβαίνουν στο HDFS με την ακόλουθη λογική: τα αρχεία των μεταδεδομένων με την κατάληξη .meta τοποθετούνται στο InputPath της MapReduce εργασίας που εκτελεί την επερώτηση με τη χρήση του R-tree ευρετηρίου, ενώ τα υπόλοιπα 7 αρχεία τοποθετούνται σε διαφορετική τοποθεσία που έχει οριστεί κατά την εγκατάσταση. Η διάκριση αυτή γίνεται γιατί δε θέλουμε η συνάρτηση map να διατρέξει όλα τα αρχεία του ευρετηρίου παρά μόνο το αρχείο των μεταδεδομένων που είναι απαραίτητο για την έναρξη της επεξεργασίας της επερώτησης. Η διαδικασία δημιουργίας των ευρετηρίων και το ανέβασμα τους στο HDFS αναπαρίσταται γραφικά ακολούθως:



Εικόνα 13: Διαχωρισμός αρχείων εισόδου και δημιουργία ευρετηρίου

6.5 MapReduce εργασία για εκτέλεση της επερώτησης με R-tree ευρετήριο

Η επεξεργασία της επερώτησης με χρήση του R-tree based ευρετηρίου γίνεται με τη μορφή MapReduce εργασίας ο κώδικας της οποίας έχει τροποποιηθεί έτσι ώστε να εκτελείται μέσω συναρτήσεων map() και reduce(). Εκτός από τις κλάσεις Mapper και Reducer που αναπτύχθηκαν υλοποιήθηκε για της εργασία αυτή και ένα ειδικά σχεδιασμένο InputFormat το οποίο επιφορτίζεται με το ρόλο της τροφοδοσίας της συνάρτησης map() με ζεύγη κλειδιού-τιμής. Ένα από τα πλεονεκτήματα της σχεδίασης αυτής είναι ότι το ευρετήριο μπορεί να χρησιμοποιηθεί και από άλλες MapReduce εργασίες χωρίς να απαιτούνται ιδιαίτερες τροποποιήσεις στις map και reduce μεθόδους τους, αρκεί να χρησιμοποιήσουν το ειδικά σχεδιασμένο InputFormat.

6.5.1 INPUT FORMAT

Το ειδικά σχεδιασμένο Input Format που αναπτύχθηκε ονομάζεται WholeFileInputFormat και παίρνει σαν είσοδο ολόκληρο το αρχείο μεταδεδομένων για κάθε ευρετήριο. Δε διαβάζει το αρχείο γραμμή γραμμή παρά ανοίγει ένα ObjectInputStream και ανακτά τα μεταδεδομένα που είναι απαραίτητα για την εκτέλεση της επερώτησης. Αφού τα ανακτήσει εκτελεί την επερώτηση και παίρνει τα αποτελέσματά της, doublePoints που ικανοποιούν τη συνθήκη της. Κάθε ένα από αυτά τα doublePoints το μετατρέπει σε string και το προσαρτά σαν νέα γραμμή στην τιμή του ζεύγους κλειδιού-τιμής που θα περάσει στη συνάρτηση map(). Σαν key ορίζει ένα τυχαίο intWritable (π.χ το 1). Επομένως για κάθε ευρετήριο περνά στη συνάρτηση map ένα ζεύγος κλειδιού –τιμής της μορφής (IntWritable, Text) όπου στην τιμή που είναι κείμενο περιλαμβάνονται όλα τα σημεία του αρχείου το οποίο ευρετηριάζει το συγκεκριμένο R-tree που ικανοποιούν τη συνθήκη της επερώτησης. Κάθε γραμμή του κειμένου αναπαριστά και ένα σημείο.

6.5.2 MAPPER

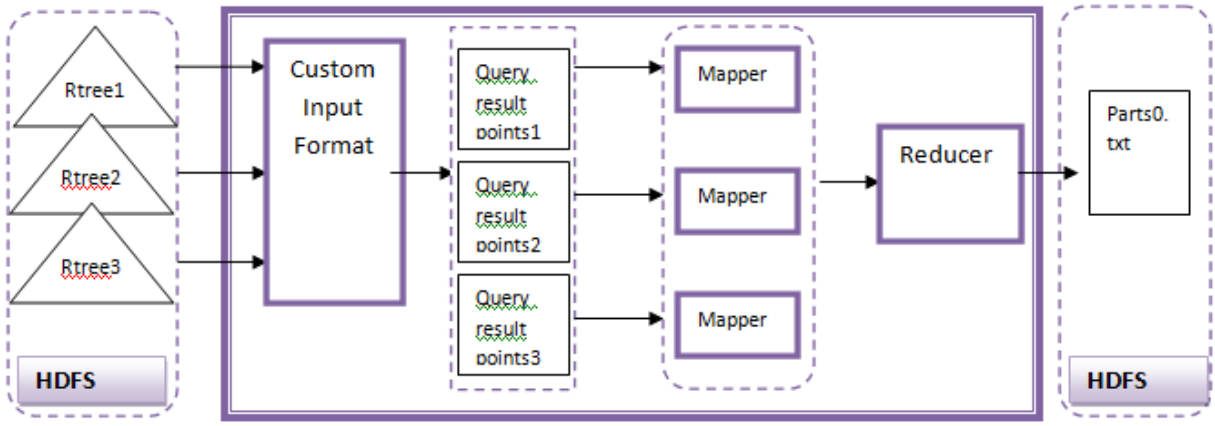
Η κλάση που υλοποιεί το Mapper για αυτό την εργασία περιλαμβάνει μια συνάρτηση map που παίρνει σαν είσοδο ζεύγη κλειδιού τιμής της μορφής (IntWritable, Text) όπως αυτά προκύπτουν από το InputFormat που αναπτύχθηκε. Κάθε τιμή είναι ένα κείμενο όπου κάθε γραμμή είναι και ένα διαφορετικό σημείο που ικανοποιεί τη συνθήκη της επερώτησης. Για κάθε ζεύγος κλειδιού-τιμής επεξεργάζεται την τιμή και ο collector συλλέγει κάθε γραμμή της σαν key με value έναν

τυχαίο IntWritable. Τα ζεύγη αυτά της μορφής (Text, IntWritable) περνάν σαν είσοδος στη συνάρτηση reduce.

6.5.3 REDUCER

Η κλάση που υλοποιεί το Reducer γι' αυτή την εργασία παίρνει σαν είσοδο ζεύγη κλειδιού-τιμής της μορφής (Text,IntWritable) όπως προκύπτουν από την map και τα συλλέγει χωρίς να κάνει κάποια επιπλέον ταξινόμηση ή επεξεργασία. Τα αποτελέσματα γράφονται στο HDFS και κάθε γραμμή έχει τη μορφή κλειδί – ολόκληρη τη γραμμή που αναπαριστά το σημείο που ικανοποιεί τη συνθήκη της επερώτησης και τιμή έναν intWritable.

Η διαδικασία εκτέλεσης της επερώτησης αναπαρίσταται γραφικά στο ακόλουθο σχήμα.



Εικόνα 14: Ροή δεδομένων κατά την εκτέλεση της επερώτησης με χρήση ευρετηρίου

6.6 MapReduce εργασία για εκτέλεση της επερώτησης χωρίς ευρετήριο

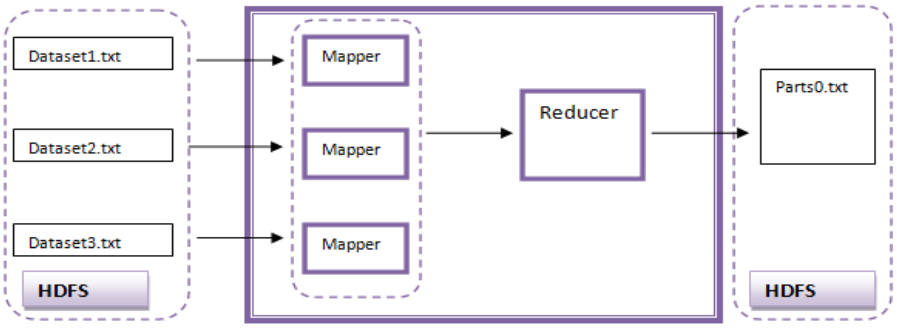
Η επεξεργασία της επερώτησης χωρίς τη χρήση R-tree ευρετηρίου πραγματοποιείται επίσης σαν MapReduce εργασία. Στην προκειμένη περίπτωση υλοποιήθηκαν μόνο κλάσεις Mapper και Reducer.

6.6.1 MAPPER

Η κλάση που υλοποιεί το Mapper για αυτή την εργασία περιλαμβάνει μια συνάρτηση map η οποία διαβάζει τα αρχεία των σημείων γραμμή γραμμή, όπου κάθε γραμμή αναπαριστά ένα σημείο. Παίρνει τις συντεταγμένες του σημείου όπως εμφανίζονται στη γραμμή του αρχείου και δημιουργεί με αυτές ένα DoublePoint. Ελέγχει αν το συγκεκριμένο DoublePoint ικανοποιεί τις συνθήκες της επερώτησης και σε περίπτωση που τις ικανοποιεί περνά τη γραμμή του αρχείου από την οποία προέκυψε το σημείο σαν κλειδί και αναθέτει σαν τιμή έναν τυχαίο IntWritable. Τα ζεύγη κλειδιού τιμής της μορφής (Text, IntWritable) περνάν σαν είσοδος στη συνάρτηση reduce.

6.6.2 REDUCER

Η κλάση που υλοποιεί το Interface Reducer γι' αυτό το job παίρνει σαν είσοδο ζεύγη κλειδιού-τιμής της μορφής (Text,IntWritable) όπως προκύπτουν από την map και τα συλλέγει χωρίς να κάνει κάποια επιπλέον ταξινόμηση η επεξεργασία. Τα αποτελέσματα γράφονται στο HDFS και κάθε γραμμή έχει τη μορφή key – ολόκληρη τη γραμμή που αναπαριστά το σημείο που ικανοποιεί τη συνθήκη της επερώτησης και value έναν intWritable.



Εικόνα 15: Ροή δεδομένων κατά την εκτέλεση της επερώτησης χωρίς χρήση ευρετηρίου

7.ΥΛΟΠΟΙΗΣΗ

Η υλοποίηση του κώδικα έγινε με τη χρήση του IDE Eclipse [38] σε γλώσσα προγραμματισμού Java με χρήση της βιβλιοθήκης του Hadoop 0.19.1 ενώ αξιοποιήθηκε το plugin του eclipse για το Hadoop για την εκτέλεση του κώδικα πάνω στο HDFS.

Οι κλάσεις που δημιουργούν το R-tree based ευρετήριο για τοπικό αρχείο καθώς και οι κλάσεις που παράγουν τα τυχαία αρχεία εισόδου με τα σημεία παραχωρήθηκαν από τον επιβλέποντα της παρούσας εργασίας κο Δουλκερίδη Χρήστο.

Η κλάση για τη δημιουργία των αρχείων εισόδου είναι η DataGenerator.java.

Οι κλάσεις για τη δημιουργία του R-tree και την εκτέλεση επερωτήσεων με την χρήση του είναι οι ακόλουθες:

- CreateRTree.java
- DataParser.java
- LeafMetaData.java
- LongLeafMetaData.java
- LoadTree.java

Οι κλάσεις αυτές χρησιμοποιήθηκαν ως έχουν για τη δημιουργία ευρετηρίων για όλα τα αρχεία εισόδου τα οποία βρίσκονται σε καθορισμένο από το χρήστη μονοπάτι στο δίσκο. Επιπλέον για τη δημιουργία ευρετηρίου και την εκτέλεση επερωτήσεων με χρήση του είναι απαραίτητη και η βιβλιοθήκη xxl. Στη συνέχεια παρουσιάζονται οι κλάσεις που υλοποιήθηκαν.

7.1 ΠΡΟΕΡΓΑΣΙΑ ΚΑΙ ΕΚΤΕΛΕΣΗ

Για την προεργασία και την εκτέλεση των MR εργασιών που εκτελούν τις επερωτήσεις υλοποιήθηκε η κλάση RangeQueryApplication.java. Στην κλάση αυτή γίνονται όλες οι παραμετροποιήσεις που αφορούν τις MR εργασίες του Hadoop και αρχικοποιούνται όλες οι μεταβλητές που χρησιμοποιούνται στον κώδικα στη συνέχεια. Ορίζεται το μονοπάτι στο οποίο δημιουργούνται τα αρχεία εισόδου και για κάθε αρχείο στο μονοπάτι αυτό με τη βοήθεια της κλάσης FileSplitter που περιγράφεται ακόλουθα συγκρίνεται το μέγεθος του με ένα ποσοστό

του HDFS blocksize και αν απαιτείται σπάει σε μικρότερα υποαρχεία ορισμένου μεγέθους. Στη συνέχεια για κάθε ένα από τα υποαρχεία που δημιουργήθηκαν με τη χρήση της κλάσης DataParser.java δημιουργείται ένα R-tree ευρετήριο που αποτελείται από 8 διαφορετικούς τύπους αρχείων. Για κάθε ένα από τα ευρετήρια που δημιουργήθηκαν ανεβαίνουν τα αρχεία που το αποτελούν στο HDFS, συγκεκριμένα το αρχείο των μεταδεδομένων κάθε ευρετηρίου ανεβαίνει στο InputPath της εργασίας που εκτελεί την επερώτηση με χρήση του R-tree ενώ τα υπόλοιπα αρχεία για κάθε ευρετήριο ανεβαίνουν στο μονοπάτι στο HDFS που έχει οριστεί στην αρχή της κλάσης. Αφού ανέβουν τα ευρετήρια στο HDFS γίνεται η παραμετροποίηση των MapReduce εργασιών. Υπάρχουν δύο διαφορετικές εργασίες και για το λόγο αυτό χρησιμοποιούνται δύο instances του JobConf για να οριστούν για καθεμία από αυτές τις εργασίες το InputPath, το OutputPath, η κλάση Mapper, η κλάση Reducer, το InputFormat καθώς και ο τύπος των τελικών κλειδιών και αξιών. Τέλος η RangeQueryApplication.java τρέχει τα 2 MR Jobs και μετρά το χρόνο εκτέλεσης τους.

7.2 Διαχωρισμός αρχείων

Για το διαμερισμό των αρχικών αρχείων εισόδου δημιουργήθηκε η κλάση FileSplitter.java. Η κλάση αυτή περιλαμβάνει δύο μεθόδους. Η μέθοδος split δέχεται σαν είσοδο ένα αρχείο και έναν long αριθμό από bytes. Αν το μέγεθος του αρχείου υπερβαίνει τον αριθμό αυτό τότε το αρχείο σπάει σε μικρότερα υποαρχεία που δεν ξεπερνούν το ορισμένο μέγεθος. Επιπλέον περιλαμβάνει και τη μέθοδο write η οποία δέχεται σαν είσοδο String και μονοπάτι προς κάποιο αρχείο και γράφει το String στο αρχείο αυτό.

7.3 Εκτέλεση επερώτησης με χρήση εγχειριδίου

Για την εκτέλεση της επερώτησης εύρους με χρήση εγχειριδίου μέσα από μια MapReduce εργασία υλοποιήθηκε η IndexQueryJob.java. Στην κλάση αυτή ορίζεται μια κλάση Mapper που επεκτείνει το interface του Mapper του Hadoop, μια κλάση Reducer που επεκτείνει το interface του Reducer , μια ειδικά σχεδιασμένη κλάση InputFormat που επεκτείνει το interface του Input Format του Hadoop καθώς και μια κλάση RecordReader για την MR εργασία που εκτελεί το επερώτηση εύρους με χρήση του R-tree based ευρετηρίου.

7.3.1 INPUTFORMAT

Η κλάση που υλοποιεί τον ειδικά σχεδιασμένο InputFormat είναι η WholeFileInputFormat και είναι εκείνη που αναλαμβάνει να διαβάσει τα αρχεία του InputPath και να τροφοδοτήσει τη συνάρτηση map με ζεύγη κλειδιού τιμής. Επεκτείνει την κλάση FileInputFormat του Hadoop και περιλαμβάνει δύο μεθόδους : τη μέθοδο isSplitable η οποία επιστρέφει ένα Boolean false που σημαίνει ότι ολόκληρο το αρχείο θα το επεξεργαστεί ένας mapper και τη μέθοδο getRecordReader η οποία επιστρέφει τον RecordReader που διαβάζει τις εγγραφές του αρχείου. Στην προκείμενη περίπτωση ο RecordReader που επιστρέφεται είναι νέο instance της κλάσης WholeFileRecordReader που παρουσιάζεται παρακάτω.

7.3.2RECORD READER

Η κλάση που υλοποίει τον RecordReader είναι η WholeFileRecordReader και είναι αυτή που διαβάζει τις εγγραφές των αρχείων του InputPath και δημιουργεί τα ζεύγη κλειδιού-τιμής που περνάνε στη συνέχεια στη συνάρτηση map. Εφαρμόζει (implements) την κλάση RecordReader του Hadoop και περιλαμβάνει τις ακόλουθες μεθόδους:

- next(LongWritable key, Text value)
- createKey()
- createValue()
- getProgress()
- close()

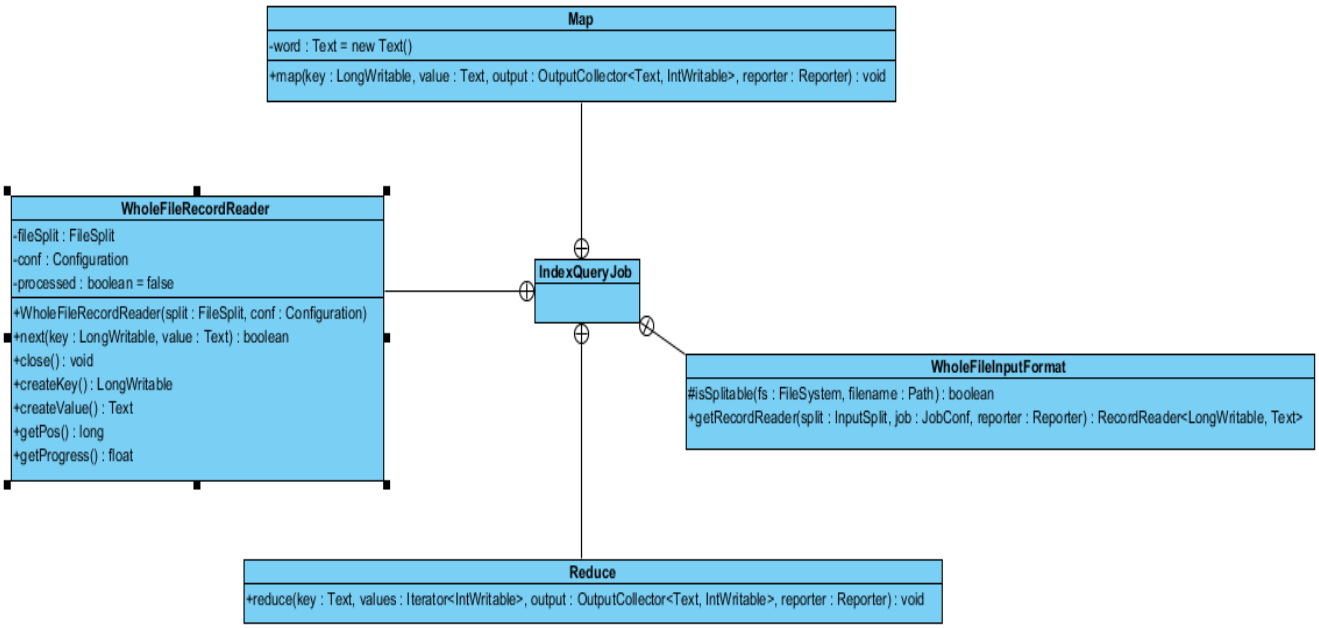
Η μέθοδος next καθορίζει τον τρόπο που θα διαβαστεί το αρχείο και δημιουργεί τα ζεύγη κλειδιού-τιμής για την επόμενη φάση της map. Δε διαβάζει ολόκληρο το αρχείο μεταδεδομένων του ευρετηρίου παρά ανοίγει ένα ObjectInputStream και ανακτά από το αρχείο τα LeafMetaData που είναι απαραίτητα για την εκτέλεση της επερώτησης. Εκτελεί την επερώτηση και προκύπτει ένα σύνολο από DoublePoints που ικανοποιούν τη συνθήκη. Κάθε ένα από αυτά τα DoublePoints το μετατρέπει σε String και το προσαρτά σαν νέα γραμμή σε ένα Text που θα αποτελέσει την τιμή στο ζεύγος κλειδιού-τιμής. Σαν κλειδί ορίζεται ένας IntWritable.

7.3.3MAPPER

Η κλάση αυτή επεκτείνει την κλάση Mapper του Hadoop και περιλαμβάνει μια συνάρτηση map η οποία δέχεται ζεύγη κλειδιού-τιμής της μορφής (IntWritable,Text) όπως προκύπτουν από το WholeFileInputFormat. Η συνάρτηση map παίρνει την τιμή της μορφής Text και για κάθε γραμμή της που αποτελεί και ένα σημείο που ικανοποιεί τη συνθήκη της επερώτησης δημιουργεί ένα νέο ζεύγος κλειδιού – τιμής με κλειδί τη γραμμή και τιμή έναν αύξοντα IntWritable. Τα ζεύγη αυτά στη συνέχεια περνάνε στη συνάρτηση Reduce.

7.3.4 REDUCER

Η κλάση αυτή επεκτείνει την κλάση Reducer του Hadoop και περιλαμβάνει μια συνάρτηση reduce η οποία δέχεται ζεύγη κλειδιού-τιμής της μορφής (Text, IntWritable) όπως προκύπτουν από το αποτέλεσμα της συνάρτησης map. Η μέθοδος reduce απλώς συλλέγει τα ζεύγη αυτά χωρίς επιπλέον ταξινόμηση ή επεξεργασία.



Εικόνα 16: IndexQueryJob class diagram

7.4 Εκτέλεση επερώτησης χωρίς χρήση εγχειριδίου

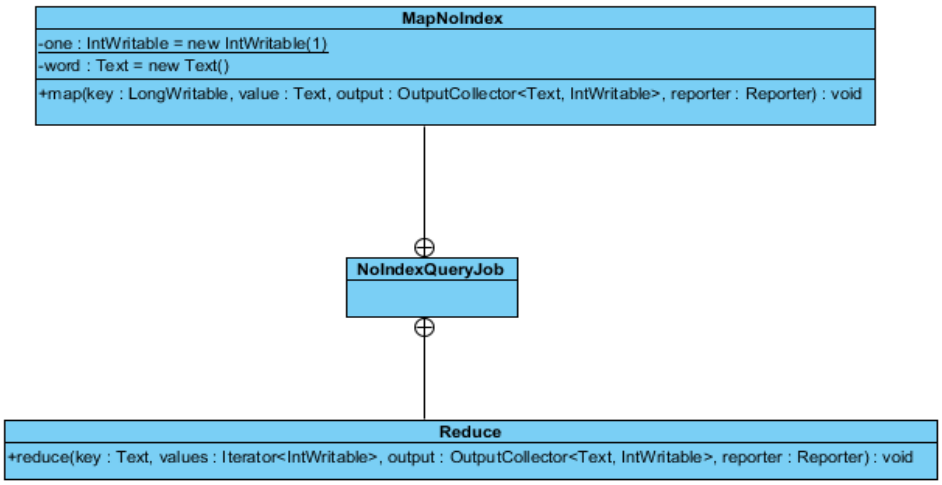
Για την εκτέλεση της επερώτησης εύρους χωρίς χρήση εγχειριδίου μέσα από μια MapReduce εργασία υλοποιήθηκε η NoIndexQueryJob.java Στην κλάση αυτή ορίζεται επίσης μια κλάση Mapper και μία κλάση Reducer για την MR εργασία που εκτελεί την επερώτηση χωρίς χρήση r-tree ευρετηρίου.

7.4.1MAPPER

Η κλάση αυτή επεκτείνει την κλάση Mapper του Hadoop και περιλαμβάνει μια συνάρτηση map η οποία δέχεται ζεύγη κλειδιού-τιμής της μορφής (IntWritable,Text). Η συνάρτηση map διαβάζει το αρχείο εισόδου γραμμή γραμμή όπου κάθε γραμμή αναπαριστά ένα σημείο. Παίρνει κάθε γραμμή του αρχείου και δημιουργεί ένα Double Point με συντεταγμένες τα στοιχεία της γραμμής. Αν το Double Point που προέκυψε ικανοποιεί τη συνθήκη της επερώτησης τότε δημιουργεί ένα νέο ζεύγος κλειδιού – τιμής με κλειδί τη γραμμή και τιμή έναν τυχαίο IntWritable. Τα ζεύγη αυτά περνούν σαν είσοδος στη συνάρτηση reduce.

7.4.2 REDUCER

Η κλάση αυτή επεκτείνει την κλάση Reducer του Hadoop και περιλαμβάνει μια συνάρτηση reduce η οποία δέχεται ζεύγη κλειδιού-τιμής της μορφής (Text, IntWritable) όπως προκύπτουν από το αποτέλεσμα της συνάρτησης map. Η μέθοδος reduce απλώς συλλέγει τα ζεύγη αυτά χωρίς επιπλέον ταξινόμηση ή επεξεργασία.



Εικόνα 17: NoIndexQueryJob class diagram

8.ΠΕΙΡΑΜΑΤΙΚΗ ΑΝΑΛΥΣΗ

Οι πειραματικές μετρήσεις πραγματοποιήθηκαν σε single node cluster σε φυσικό μηχάνημα με εγκατεστημένο το Hadoop 0.19 και τα ακόλουθα χαρακτηριστικά :

- Λειτουργικό σύστημα: Windows 7 Professional 32bit
- Μνήμη: 4 GB
- Επεξεργαστής: Intel Core Duo T5750 @ 2GHz
- Πυρήνες: 2

Η ανάπτυξη του κώδικα έγινε με χρήση του IDE Eclipse και χρησιμοποιήθηκε το plugin του Hadoop για την εκτέλεση του στο cluster. Επίσης χρησιμοποιήθηκε το Cygwin για την μετάφραση των Unix εντολών προκειμένου να ξεκινήσει η συστάδα του Hadoop.

Για την πειραματική ανάλυση επιλέχθηκε να μετρηθεί ο χρόνος σε ms για την εκτέλεση μιας επερώτησης δηλαδή για την ολοκλήρωση της MapReduce εργασίας που την εκτελεί, το μέγεθος των δεδομένων εισόδου της συνάρτησης map σε Bytes καθώς και ο αριθμός των bytes που διαβάστηκαν από το HDFS όπως μετρώνται κατά την εκτέλεση επερωτήσεων με χρήση R-tree ευρετηρίου και χωρίς τη χρήση αυτού.

Ο αριθμός των bytes εισόδου στη συνάρτηση Map μετρά τον αριθμό των bytes που ο RecordReader επεξεργάστηκε για να στείλει ζεύγη κλειδιού – τιμής στη συνάρτηση map. Αντίστοιχα τα bytes που διαβάστηκαν από το HDFS είναι τα bytes από το FS read. Στην περίπτωση που δε χρησιμοποιείται ευρετήριο τα δύο αυτά μεγέθη είναι περίπου ισοδύναμα και ισούνται με το μέγεθος του αρχείου εισόδου. Στην περίπτωση χρήσης ευρετηρίου τα bytes εισόδου της Map είναι σημαντικά μικρότερα καθώς αντιστοιχούν στο μέγεθος των αρχείων μεταδεδομένων που επεξεργάζεται από ο RecordReader ενώ τα bytes που διαβάστηκαν από το HDFS περιλαμβάνουν και το μέγεθος των δεδομένων που διαβάστηκαν από το HDFS κατά την εκτέλεση της επερώτησης.

Μετρήθηκαν τα αποτελέσματα σε αρχεία με 1.000, 10.000, 100.000, 1.000.000 και 10.000.000 σημεία με 4 και 6 διαστάσεις αντίστοιχα, δημιουργήθηκαν δηλαδή 8 διαφορετικά αρχεία εισόδου. Σε όλα τα αρχεία εισόδου επιλέχθηκε uniform κατανομή. Επιπλέον εκτελέστηκαν για κάθε περίπτωση δύο διαφορετικές επερωτήσεις που επιστρέφουν διαφορετικό πλήθος αποτελεσμάτων ώστε να μελετηθεί και η επίδραση που έχει η επιλεκτικότητα της επερώτησης στα αποτελέσματα. Για κάθε περίπτωση

παρουσιάζεται ένας πίνακας αποτελεσμάτων και έχουν σχεδιαστεί συγκριτικά γραφήματα για τα μετρήσιμα μεγέθη.

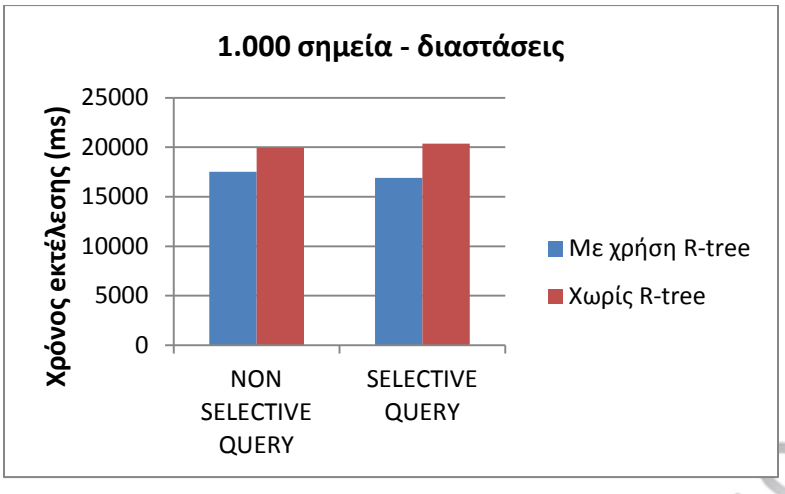
8.1 ΑΡΧΕΙΟ ΕΙΣΟΔΟΥ 1.000 ΣΗΜΕΙΩΝ

Εκτελέσαμε τον κώδικα σε αρχείο 1000 σημείων με 4 και 6 διαστάσεις και εκτελέσαμε δύο διαφορετικές επερωτήσεις ανά περίπτωση, που επιστρέφουν διαφορετικό αριθμό αποτελεσμάτων. Οι μετρήσεις δείχνουν ότι σε όλες τις περιπτώσεις η εκτέλεση της επερώτησης με χρήση ευρετηρίου είναι ταχύτερη ενώ είναι πολύ μεγάλη η διαφορά και στο bytes του map input. Στην περίπτωση της χρήσης του r-tree το map-input περιορίζεται σε μερικά bytes όσο είναι και το μέγεθος του αρχείου μεταδεδομένων του ευρετηρίου. Αντίστοιχα αν δεν χρησιμοποιείται ευρετήριο το map input ισούται με το μέγεθος του αρχείου εισόδου. Όσον αφορά τον αριθμό των bytes που διαβάζονται από το HDFS σε γενικές γραμμές είναι λιγότερα όταν χρησιμοποιείται το R-tree ευρετήριο εξαρτώνται όμως από την επιλεκτικότητα της επερώτησης. Όσο πιο επιλεκτική είναι η επερώτηση και όσο λιγότερα αποτελέσματα επιστρέφει τόσο λιγότερα bytes διαβάζονται από το HDFS όταν χρησιμοποιείται ευρετήριο. Αντίστοιχα όταν δεν χρησιμοποιείται ευρετήριο τα bytes που διαβάζονται από το HDFS είναι της τάξης του μεγέθους του αρχείου.

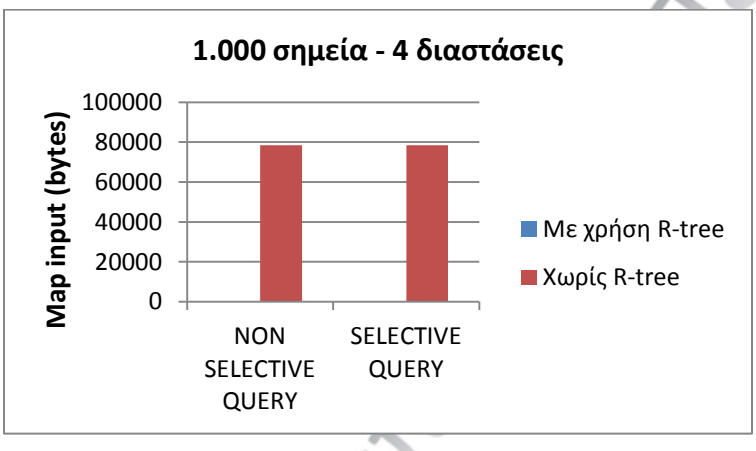
Ακολούθως ο πίνακας αποτελεσμάτων για αρχείο με σημεία 4 διαστάσεων. Το μέγεθος του αρχικού αρχείου σε bytes είναι 78,514 bytes (~76.6 KB) . Η πρώτη επερώτηση που εκτελέστηκε επέστρεψε 435 αποτελέσματα ενώ η δεύτερη 2.

1000 ΣΗΜΕΙΑ D=4	435 ΑΠΟΤΕΛΕΣΜΑΤΑ		2 ΑΠΟΤΕΛΕΣΜΑΤΑ	
	R-tree	NO INDEX	R-tree	NO INDEX
TIME IN ms	20638	24020	17510	19933
MAP INPUT IN bytes	273	78514	273	78514
HDFS BYTES READ	99091	80218	17161	80218

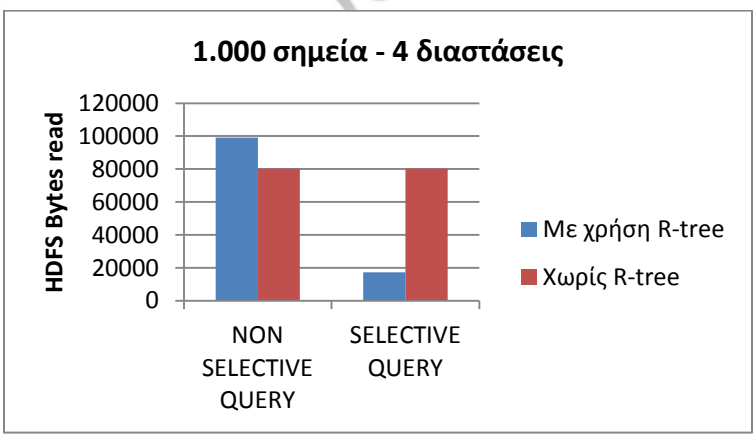
Ακολουθούν τα συγκριτικά διαγράμματα για το χρόνο εκτέλεσης της επερώτησης, τα map input bytes και τα bytes που διαβάστηκαν από το HDFS για σημεία 4 διαστάσεων.



Εικόνα 18 : Χρόνος εκτέλεσης επερώτησης σε αρχείο 1000 σημείων 4 διαστάσεων



Εικόνα 19: Bytes εισόδου στη συνάρτηση map για αρχείο 1000 σημείων 4 διαστάσεων

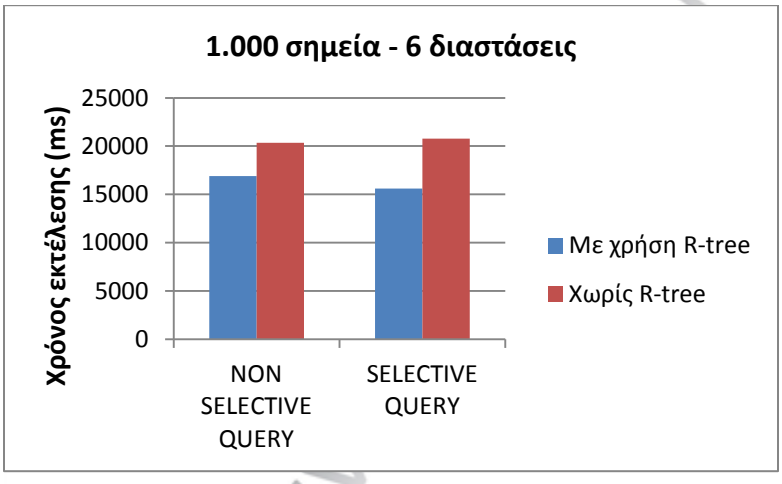


Εικόνα 20: Bytes που διαβάστηκαν από το HDFS για αρχείο 1000 σημείων 4 διαστάσεων

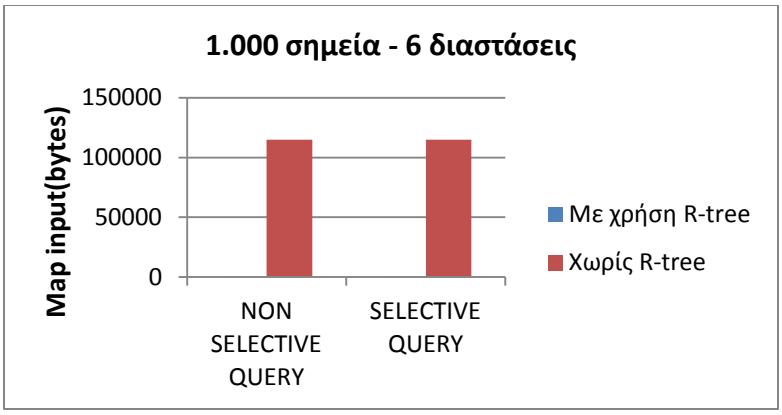
Ακολούθως ο πίνακας αποτελεσμάτων για αρχείο με σημεία 6 διαστάσεων. Το μέγεθος του αρχικού αρχείου σε bytes είναι 114,827 bytes (~112 KB). Η πρώτη επερώτηση που εκτελέστηκε επέστρεψε 393 αποτελέσματα ενώ η δεύτερη 2.

1000 ΣΗΜΕΙΑ D=6	393 αποτελέσματα		2 αποτελέσματα	
	R-tree	NO INDEX	R-tree	NO INDEX
TIME IN ms	16890	20354	15622	20768
MAP INPUT IN bytes	273	114827	273	114827
HDFS BYTES READ	115541	118855	49997	118855

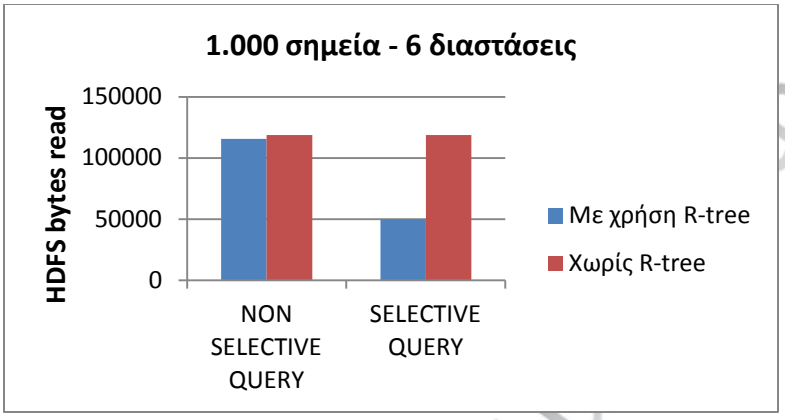
Ακολουθούν τα συγκριτικά διαγράμματα για το χρόνο εκτέλεσης της επερώτησης, τα map input bytes και τα bytes που διαβάστηκαν από το HDFS για σημεία 6 διαστάσεων.



Εικόνα 21: Χρόνος εκτέλεσης επερώτησης σε αρχείο 1000 σημείων 6 διαστάσεων



Εικόνα 22: Bytes εισόδου στη συνάρτηση map για αρχείο 1000 σημείων 6 διαστάσεων



Εικόνα 23: Bytes που διαβάστηκαν από το HDFS για αρχείο 1000 σημείων 6 διαστάσεων

8.2 ΑΡΧΕΙΟ ΕΙΣΟΔΟΥ 10.000 ΣΗΜΕΙΩΝ

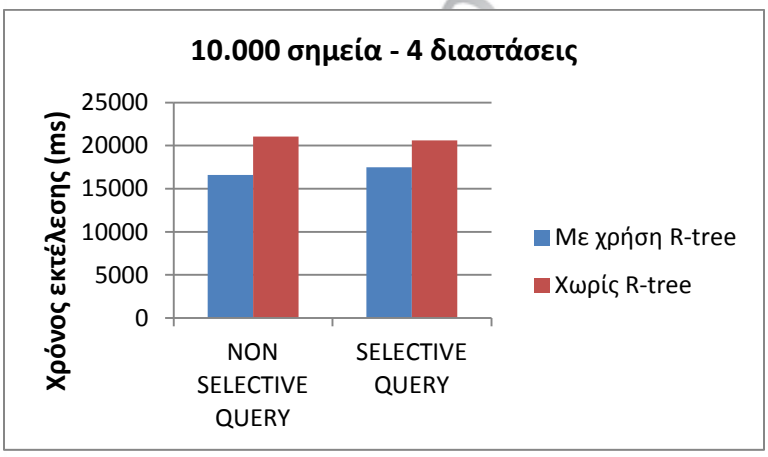
Εκτελέσαμε τον κώδικα σε αρχείο 10000 σημείων με 4 και 6 διαστάσεις και εκτελέσαμε δύο διαφορετικές επερωτήσεις ανά περίπτωση, που επιστρέφουν διαφορετικό αριθμό αποτελεσμάτων. Οι μετρήσεις δείχνουν ότι σε όλες τις περιπτώσεις η εκτέλεση της επερώτησης με χρήση ευρετηρίου είναι ταχύτερη ενώ είναι πολύ μεγάλη η διαφορά και στο bytes του map input. Στην περίπτωση της χρήσης του r-tree το map-input περιορίζεται σε μερικά bytes όσο είναι και το μέγεθος του αρχείου μεταδεδομένων του ευρετηρίου. Αντίστοιχα αν δεν χρησιμοποιείται ευρετήριο το map input ισούται με το μέγεθος του αρχείου εισόδου. Όσον αφορά τον αριθμό των bytes που διαβάζονται από το HDFS σε γενικές γραμμές είναι λιγότερα όταν χρησιμοποιείται το r-tree ευρετήριο εξαρτώνται όμως από την επιλεκτικότητα της επερώτησης. Όσο πιο επιλεκτική είναι η επερώτηση και όσο λιγότερα αποτελέσματα επιστρέφει

τόσο λιγότερα bytes διαβάζονται από το HDFS όταν χρησιμοποιείται ευρετήριο. Αντίστοιχα όταν δεν χρησιμοποιείται ευρετήριο τα bytes που διαβάζονται από το HDFS είναι της τάξης του μεγέθους του αρχείου.

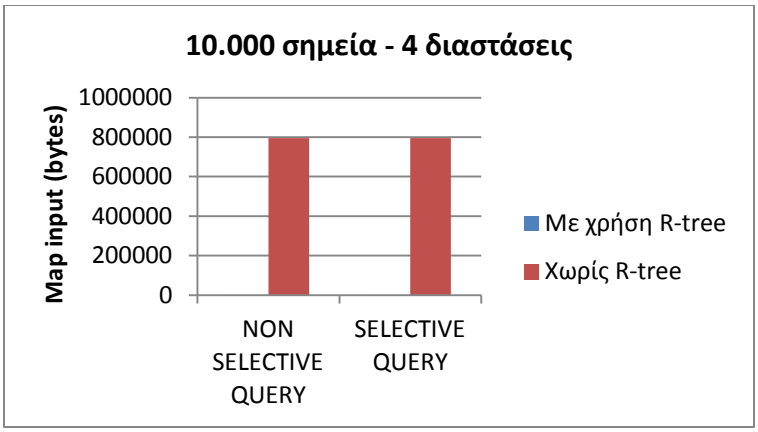
Ακολούθως ο πίνακας αποτελεσμάτων για αρχείο με σημεία 4 διαστάσεων. Το μέγεθος του αρχικού αρχείου σε bytes είναι 795,249 bytes (~776 KB). Η πρώτη επερώτηση που εκτελέστηκε επέστρεψε 469 αποτελέσματα ενώ η δεύτερη 8.

10000 ΣΗΜΕΙΑ D=4	469 αποτελέσματα		8 αποτελέσματα	
	R-tree	NO INDEX	R-tree	NO INDEX
TIME IN ms	16599	21059	17479	20619
MAP INPUT IN bytes	273	795249	273	795249
HDFS BYTES READ	213793	799034	25354	799034

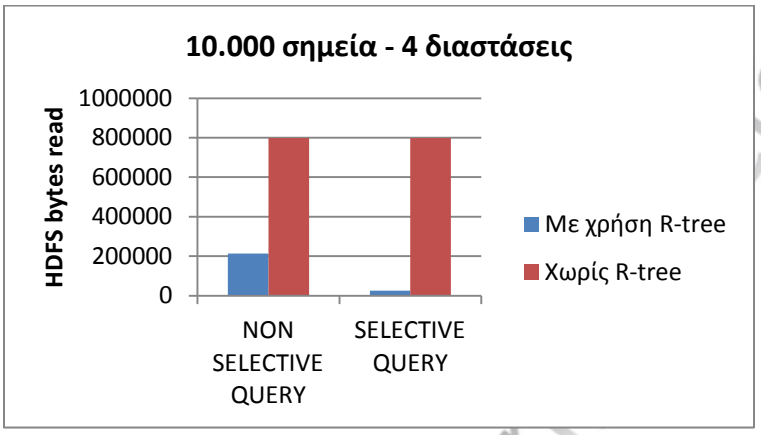
Ακολουθούν τα συγκριτικά διαγράμματα για το χρόνο εκτέλεσης της επερώτησης, τα map input bytes και τα bytes που διαβάστηκαν από το HDFS για σημεία 4 διαστάσεων.



Εικόνα 24: Χρόνος εκτέλεσης επερώτησης σε αρχείο 10.000 σημείων 4 διαστάσεων



Εικόνα 25: Bytes εισόδου στη συνάρτηση map για αρχείο 10.000 σημείων 4 διαστάσεων

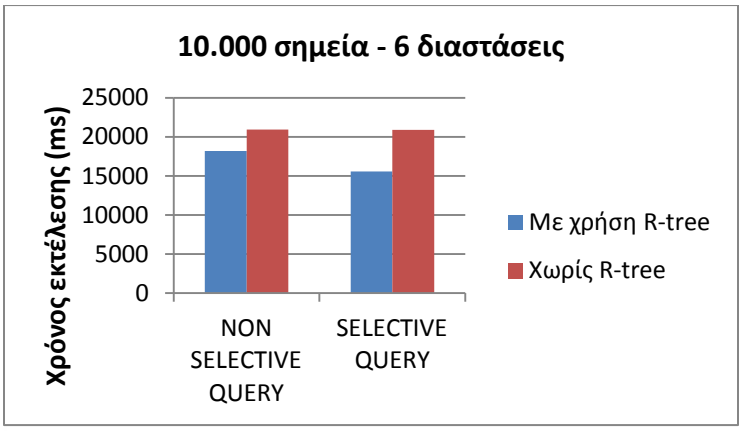


Εικόνα 26: Bytes που διαβάστηκαν από το HDFS για αρχείο 10.000 σημείων 4 διαστάσεων

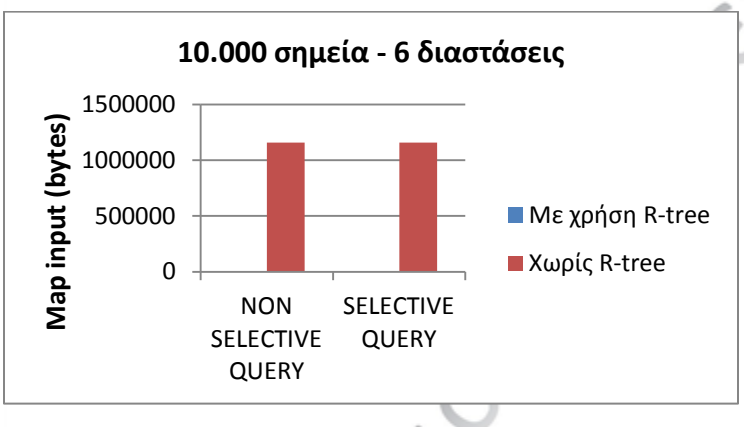
Ακολούθως ο πίνακας αποτελεσμάτων για αρχείο με σημεία 6 διαστάσεων. Το μέγεθος του αρχικού αρχείου σε bytes είναι 1,158,492 bytes (~1.10 MB). Η πρώτη επερώτηση που εκτελέστηκε επέστρεψε 267 αποτελέσματα ενώ η δεύτερη 8.

10000 ΣΗΜΕΙΑ D=6	267 αποτελέσματα		8 αποτελέσματα	
	R-tree	NO INDEX	R-tree	NO INDEX
TIME IN ms	18170	20936	15563	20886
MAP INPUT IN bytes	273	1158492	273	1158492
HDFS BYTES READ	689051	1160879	49997	1160879

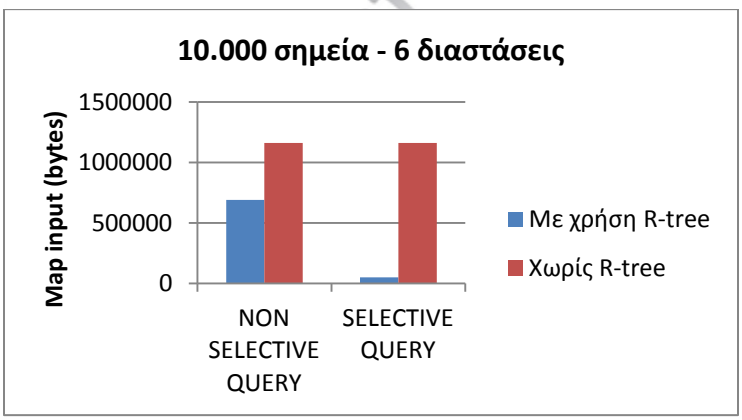
Ακολουθούν τα συγκριτικά διαγράμματα για το χρόνο εκτέλεσης της επερώτησης, τα map input bytes και τα bytes που διαβάστηκαν από το HDFS για σημεία 6 διαστάσεων.



Εικόνα 27: Χρόνος εκτέλεσης επερώτησης σε αρχείο 10.000 σημείων 6 διαστάσεων



Εικόνα 28: Bytes εισόδου στη συνάρτηση map για αρχείο 10.000 σημείων 6 διαστάσεων



Εικόνα 29: Bytes που διαβάστηκαν από το HDFS για αρχείο 10.000 σημείων 6 διαστάσεων

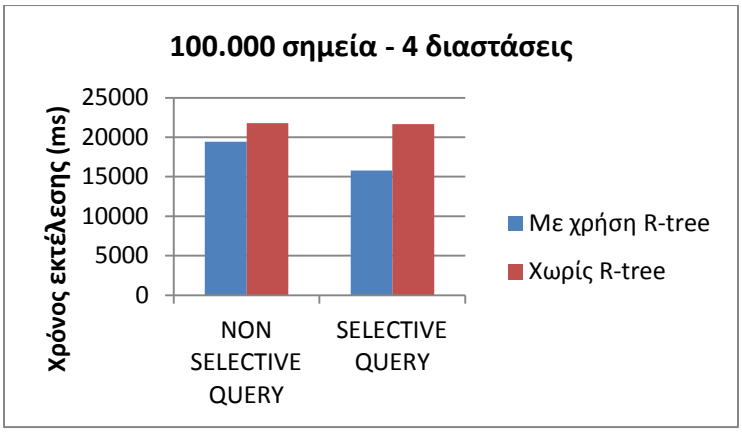
8.3 ΑΡΧΕΙΟ ΕΙΣΟΔΟΥ 100.000 ΣΗΜΕΙΩΝ

Εκτελέσαμε τον κώδικα σε αρχείο 100.000 σημείων με 4 και 6 διαστάσεις και εκτελέσαμε δύο διαφορετικές επερωτήσεις ανά περίπτωση, που επιστρέφουν διαφορετικό αριθμό αποτελεσμάτων. Οι μετρήσεις δείχνουν ότι σε όλες τις περιπτώσεις η εκτέλεση της επερώτησης με χρήση ευρετηρίου είναι ταχύτερη ενώ είναι πολύ μεγάλη η διαφορά και στο bytes του map input. Στην περίπτωση της χρήσης του r-tree το map-input περιορίζεται σε μερικά bytes όσο είναι και το μέγεθος του αρχείου μεταδεδομένων του ευρετηρίου. Αντίστοιχα αν δεν χρησιμοποιείται ευρετήριο το map input ισούται με το μέγεθος του αρχείου εισόδου. Όσον αφορά τον αριθμό των bytes που διαβάζονται από το HDFS σε γενικές γραμμές είναι λιγότερα όταν χρησιμοποιείται το r-tree ευρετήριο εξαρτώνται όμως από την επιλεκτικότητα της επερώτησης. Όσο πιο επιλεκτική είναι η επερώτηση και όσο λιγότερα αποτελέσματα επιστρέφει τόσο λιγότερα bytes διαβάζονται από το HDFS όταν χρησιμοποιείται ευρετήριο. Αντίστοιχα όταν δεν χρησιμοποιείται ευρετήριο τα bytes που διαβάζονται από το HDFS είναι της τάξης του μεγέθους του αρχείου.

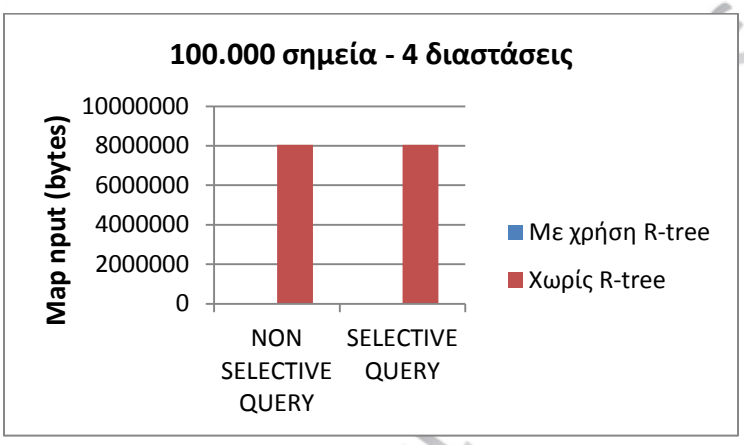
Ακολούθως ο πίνακας αποτελεσμάτων για αρχείο με σημεία 4 διαστάσεων. Το μέγεθος του αρχικού αρχείου σε bytes είναι 8,051,377 bytes (~7.67 MB). Η πρώτη επερώτηση που εκτελέστηκε επέστρεψε 644 αποτελέσματα ενώ η δεύτερη 10.

100000 ΣΗΜΕΙΑ D=4	644 αποτελέσματα		10 αποτελέσματα	
	R-tree	NO INDEX	R-tree	NO INDEX
TIME IN ms	19445	21792	15784	21667
MAP INPUT IN bytes	273	8051377	273	8051377
HDFS BYTES READ	484162	8052058	25354	8052058

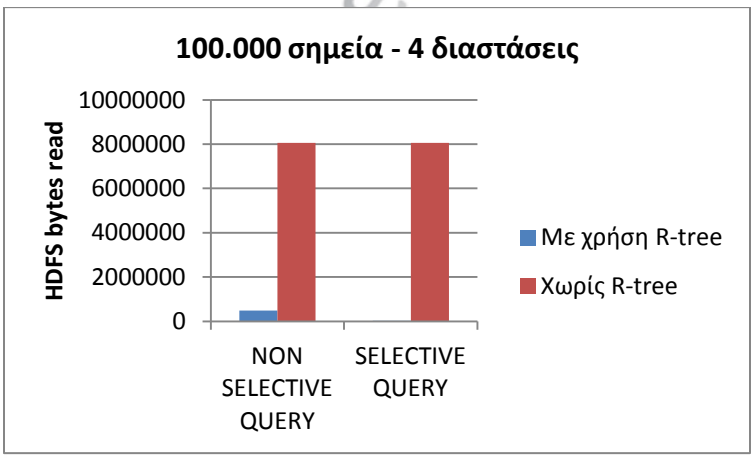
Ακολουθούν τα συγκριτικά διαγράμματα για το χρόνο εκτέλεσης της επερώτησης, τα map input bytes και τα bytes που διαβάστηκαν από το HDFS για σημεία 4 διαστάσεων.



Εικόνα 30: Χρόνος εκτέλεσης επερώτησης σε αρχείο 100.000 σημείων 4 διαστάσεων



Εικόνα 31: Bytes εισόδου στη συνάρτηση map για αρχείο 100.000 σημείων 4 διαστάσεων

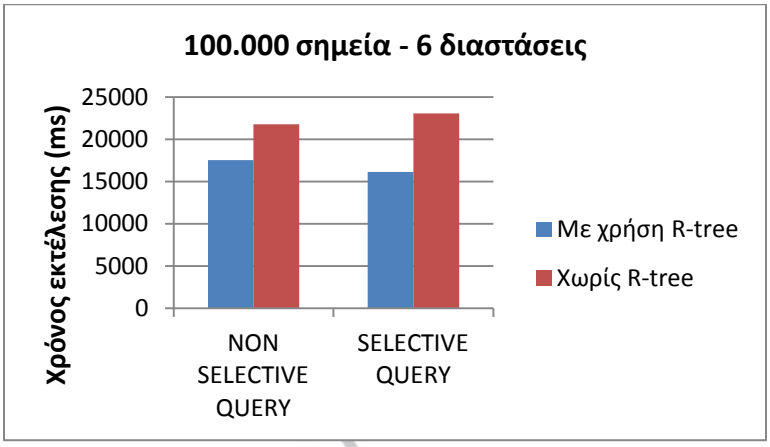


Εικόνα 32: Bytes που διαβάστηκαν από το HDFS για αρχείο 100.000 σημείων 4 διαστάσεων

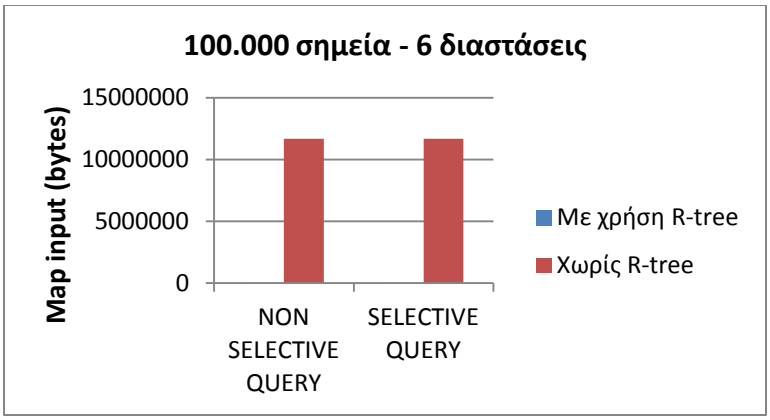
Ακολούθως ο πίνακας αποτελεσμάτων για αρχείο με σημεία 6 διαστάσεων. Το μέγεθος του αρχικού αρχείου σε bytes είναι 11,682,081 bytes (~11.1 MB). Η πρώτη επερώτηση που εκτελέστηκε επέστρεψε 1514 αποτελέσματα ενώ η δεύτερη 2.

100000 ΣΗΜΕΙΑ D=6	1514 αποτελέσματα		2 αποτελέσματα	
	R-tree	NO INDEX	R-tree	NO INDEX
TIME IN ms	17547	21775	16141	23058
MAP INPUT IN bytes	273	11682081	273	11682081
HDFS BYTES READ	1606667	11686034	33611	11686034

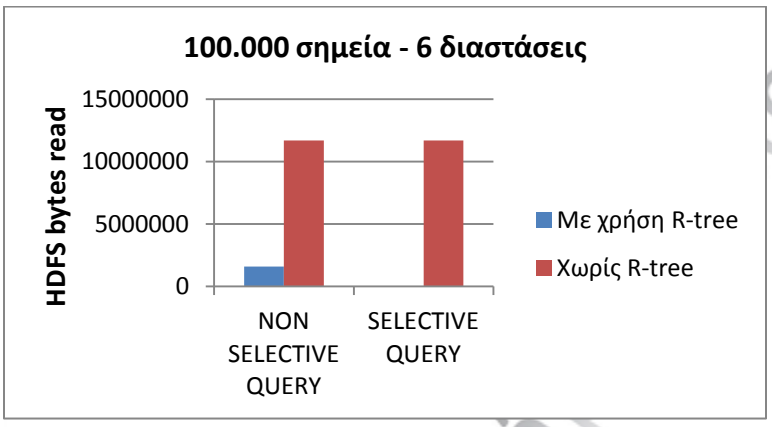
Ακολουθούν τα συγκριτικά διαγράμματα για το χρόνο εκτέλεσης της επερώτησης, τα map input bytes και τα bytes που διαβάστηκαν από το HDFS για σημεία 6 διαστάσεων.



Εικόνα 33: Χρόνος εκτέλεσης επερώτησης σε αρχείο 100.00 σημείων 6 διαστάσεων



Εικόνα 34: Bytes εισόδου στη συνάρτηση map για αρχείο 100.000 σημείων 6 διαστάσεων



Εικόνα 35: Bytes που διαβάστηκαν από το HDFS για αρχείο 100.000 σημείων 6 διαστάσεων

8.4 ΑΡΧΕΙΟ ΕΙΣΟΔΟΥ 1.000.000 ΣΗΜΕΙΩΝ

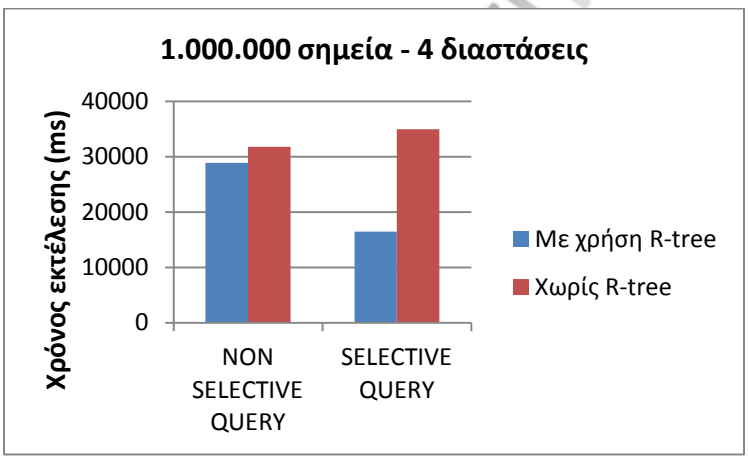
Εκτελέσαμε τον κώδικα σε αρχείο 1.000.000 σημείων με 4 και 6 διαστάσεις και εκτελέσαμε δύο διαφορετικές επερωτήσεις, που επιστρέφουν διαφορετικό αριθμό αποτελεσμάτων. Οι μετρήσεις δείχνουν ότι σε όλες τις περιπτώσεις η εκτέλεση της επερώτησης με χρήση ευρετηρίου είναι ταχύτερη ενώ είναι πολύ μεγάλη η διαφορά και στο bytes του map input. Στην περίπτωση της χρήσης του r-tree το map-input περιορίζεται σε μερικά bytes όσο είναι και το μέγεθος του αρχείου μεταδεδομένων του ευρετηρίου. Αντίστοιχα αν δεν χρησιμοποιείται ευρετήριο το map input ισούται με το μέγεθος του αρχείου εισόδου. Όσον αφορά τον αριθμό των bytes που διαβάζονται από το HDFS σε γενικές γραμμές είναι λιγότερα όταν χρησιμοποιείται το r-tree ευρετήριο εξαρτώνται όμως από την επιλεκτικότητα της επερώτησης. Όσο πιο επιλεκτική είναι η

επερώτηση και όσο λιγότερα αποτελέσματα επιστρέφει τόσο λιγότερα bytes διαβάζονται από το HDFS όταν χρησιμοποιείται ευρετήριο. Αντίστοιχα όταν δεν χρησιμοποιείται ευρετήριο τα bytes που διαβάζονται από το HDFS είναι της τάξης του μεγέθους του αρχείου.

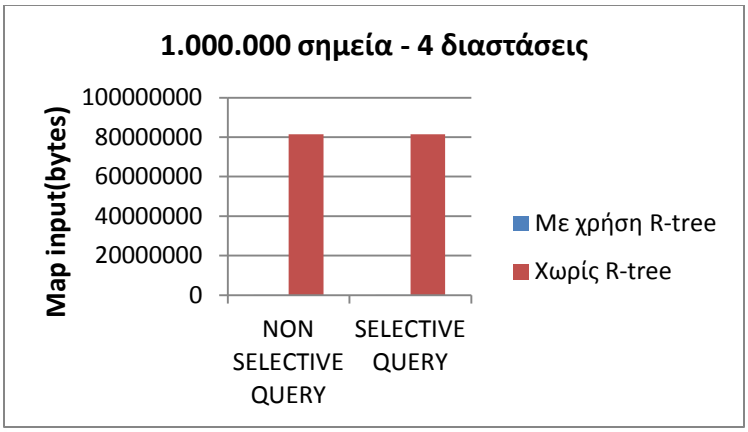
Ακολούθως ο πίνακας αποτελεσμάτων για αρχείο με σημεία 4 διαστάσεων. Το μέγεθος του αρχικού αρχείου σε bytes είναι 81,538,777 bytes (~77.7 MB). Η πρώτη επερώτηση που εκτελέστηκε επέστρεψε 4754 αποτελέσματα ενώ η δεύτερη 4.

1000000 ΣΗΜΕΙΑ D=4	4754 αποτελέσματα		4 αποτελέσματα	
	R-tree	NO INDEX	R-tree	NO INDEX
TIME IN ms	28897	31764	16454	34964
MAP INPUT IN bytes	273	81538777	273	81538777
HDFS BYTES READ	1590217	81540974	58126	81540974

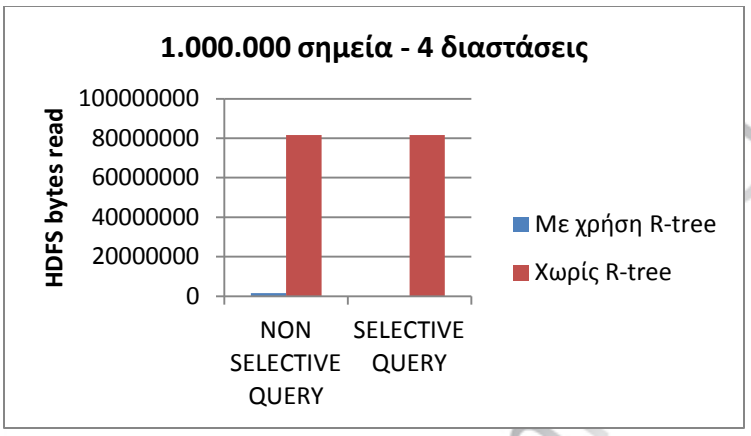
Ακολουθούν τα συγκριτικά διαγράμματα για το χρόνο εκτέλεσης της επερώτησης, τα map input bytes και τα bytes που διαβάστηκαν από το HDFS για σημεία 4 διαστάσεων.



Εικόνα 36: Χρόνος εκτέλεσης επερώτησης σε αρχείο 1.000.000 σημείων 4 διαστάσεων



Εικόνα 37: Bytes εισόδου στη συνάρτηση map για αρχείο 1.000.000 σημείων 4 διαστάσεων

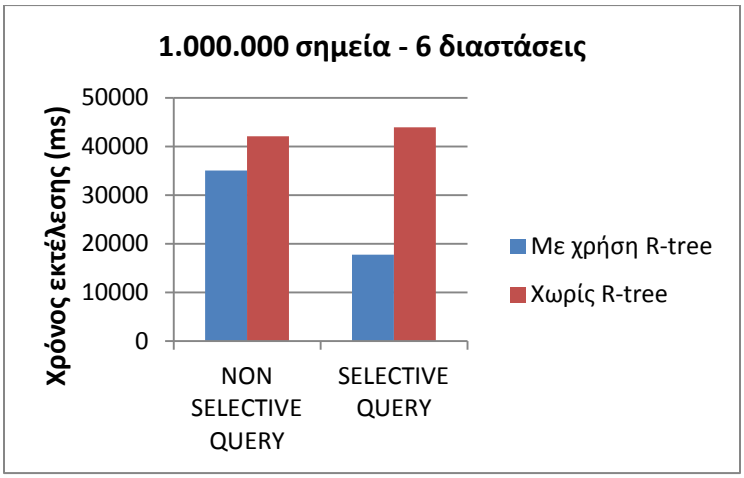


Εικόνα 38: Bytes που διαβάστηκαν από το HDFS για αρχείο 1.000.000 σημείων 4 διαστάσεων

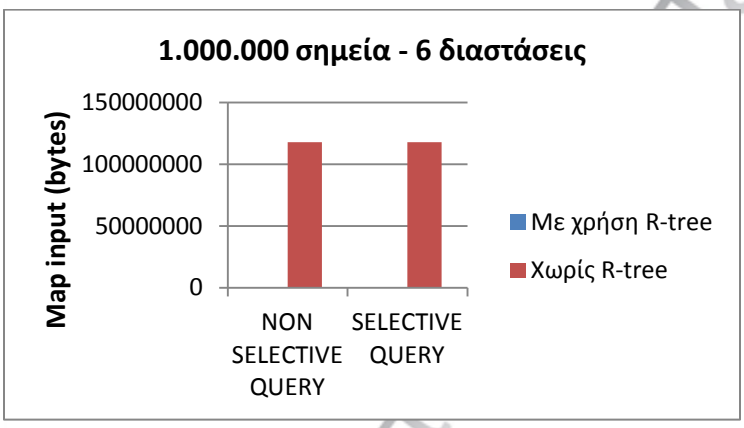
Ακολούθως ο πίνακας αποτελεσμάτων για αρχείο με σημεία 6 διαστάσεων. Το μέγεθος του αρχικού αρχείου σε bytes είναι 117,861,945 bytes (~112 MB). Η πρώτη επερώτηση που εκτελέστηκε επέστρεψε 2119 αποτελέσματα ενώ η δεύτερη 2.

1000000 ΣΗΜΕΙΑ D=6	2119 αποτελέσματα		2 αποτελέσματα	
	R-tree	NO INDEX	R-tree	NO INDEX
TIME IN ms	35043	42104	17782	43910
MAP INPUT IN bytes	273	117861945	273	117861945
HDFS BYTES READ	4523375	117864222	58190	117864222

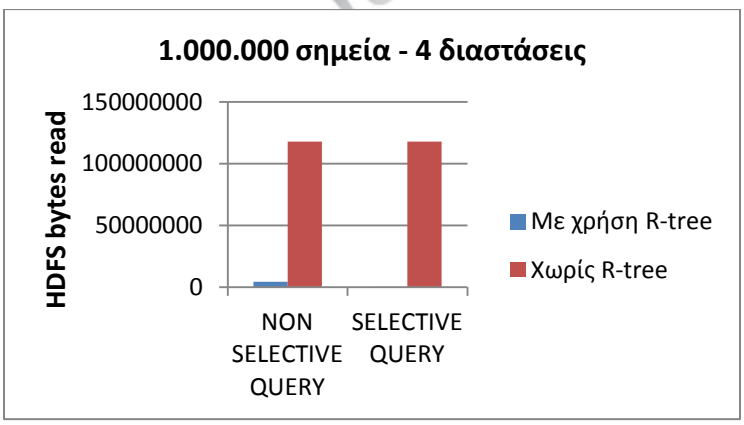
Ακολουθούν τα συγκριτικά διαγράμματα για το χρόνο εκτέλεσης της επερώτησης, τα map input bytes και τα bytes που διαβάστηκαν από το HDFS για σημεία 6 διαστάσεων.



Εικόνα 39: Χρόνος εκτέλεσης επερώτησης σε αρχείο 1.000.0000 σημείων 6 διαστάσεων



Εικόνα 40: Bytes εισόδου στη συνάρτηση map για αρχείο 1.000.000 σημείων 6 διαστάσεων



Εικόνα 41: Bytes που διαβάστηκαν από το HDFS για αρχείο 1.000.000 σημείων 6 διαστάσεων

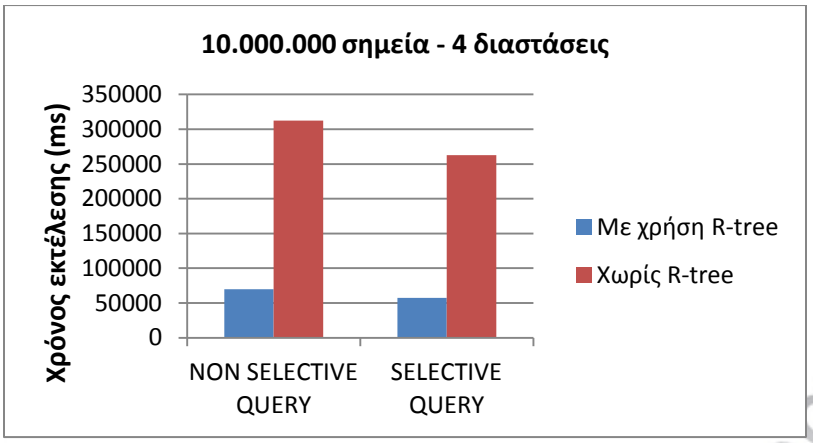
8.5 ΑΡΧΕΙΟ ΕΙΣΟΔΟΥ 10.000.000 ΣΗΜΕΙΩΝ

Εκτελέσαμε τον κώδικα σε αρχείο 10.000.000 σημείων με 4 και 6 διαστάσεις και εκτελέσαμε δύο διαφορετικές επερωτήσεις, που επιστρέφουν διαφορετικό αριθμό αποτελεσμάτων. Οι μετρήσεις δείχνουν ότι σε όλες τις περιπτώσεις η εκτέλεση της επερώτησης με χρήση ευρετηρίου είναι ταχύτερη ενώ είναι πολύ μεγάλη η διαφορά και στο bytes του map input. Στην περίπτωση της χρήσης του r-tree το map-input περιορίζεται σε μερικά bytes όσο είναι και το μέγεθος του αρχείου μεταδεδομένων του ευρετηρίου. Αντίστοιχα αν δεν χρησιμοποιείται ευρετήριο το map input ισούται με το μέγεθος του αρχείου εισόδου. Όσον αφορά τον αριθμό των bytes που διαβάζονται από το HDFS σε γενικές γραμμές είναι λιγότερα όταν χρησιμοποιείται το R-tree ευρετήριο εξαρτώνται όμως από την επιλεκτικότητα της επερώτησης. Όσο πιο επιλεκτική είναι η επερώτηση και όσο λιγότερα αποτελέσματα επιστρέφει τόσο λιγότερα bytes διαβάζονται από το HDFS όταν χρησιμοποιείται ευρετήριο. Αντίστοιχα όταν δεν χρησιμοποιείται ευρετήριο τα bytes που διαβάζονται από το HDFS είναι της τάξης του μεγέθους του αρχείου.

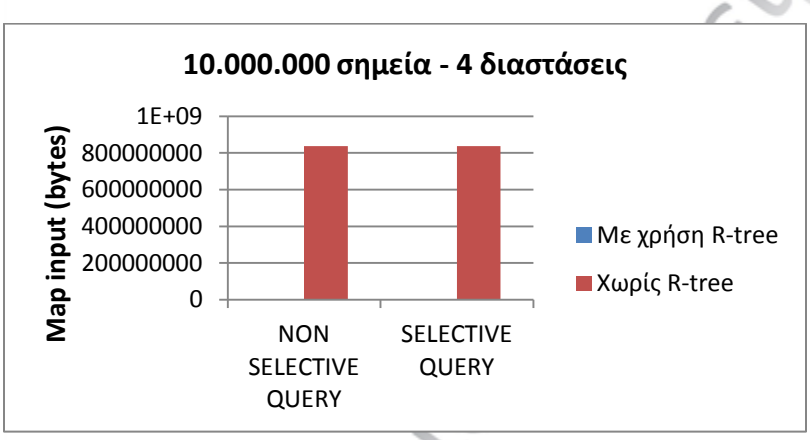
Ακολούθως ο πίνακας αποτελεσμάτων για αρχείο με σημεία 4 διαστάσεων. Το μέγεθος του αρχικού αρχείου σε bytes είναι 837,130,225 bytes (~798MB). Η πρώτη επερώτηση που εκτελέστηκε επέστρεψε 10643 αποτελέσματα ενώ η δεύτερη 7.

10000000 ΣΗΜΕΙΑ D=4	10643 αποτελέσματα		7 αποτελέσματα	
	R-tree	NO INDEX	R-tree	NO INDEX
TIME IN ms	69746	312503	57359	262583
MAP INPUT IN bytes	273	837130225	273	837130225
HDFS BYTES READ	3023992	837179389	58126	837179389

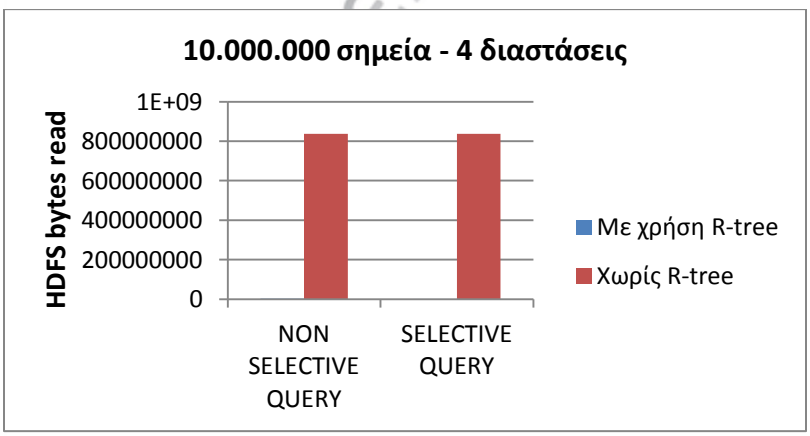
Ακολουθούν τα συγκριτικά διαγράμματα για το χρόνο εκτέλεσης της επερώτησης, τα map input bytes και τα bytes που διαβάστηκαν από το HDFS για σημεία 4 διαστάσεων.



Εικόνα 42: Χρόνος εκτέλεσης επερώτησης σε αρχείο 10.000.000 σημείων 4 διαστάσεων



Εικόνα 43: Bytes εισόδου στη συνάρτηση map για αρχείο 10.000.000 σημείων 4 διαστάσεων

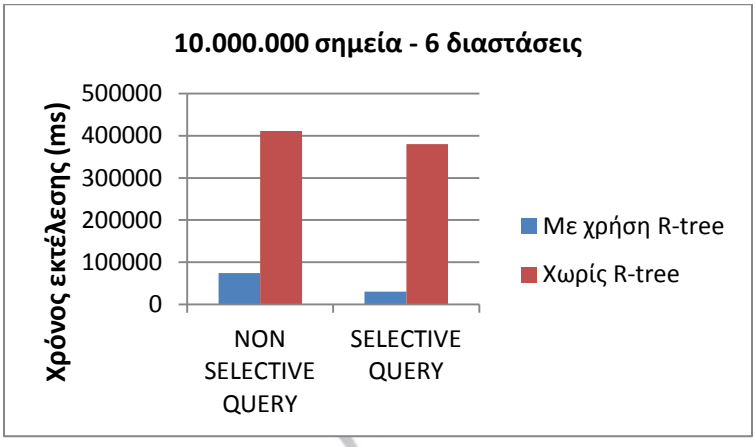


Εικόνα 44: Bytes που διαβάστηκαν από το HDFS για αρχείο 10.000.000 σημείων 4 διαστάσεων

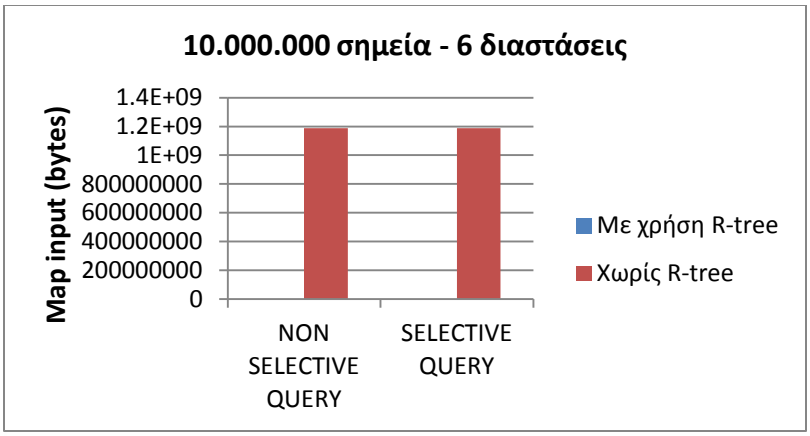
Ακολούθως ο πίνακας αποτελεσμάτων για αρχείο με σημεία 6 διαστάσεων. Το μέγεθος του αρχικού αρχείου σε bytes είναι 1,188,630,739 bytes (~1.1GB). Η πρώτη επερώτηση που εκτελέστηκε επέστρεψε 2986 αποτελέσματα ενώ η δεύτερη 7.

10000000 ΣΗΜΕΙΑ D=6	2986 αποτελέσματα		7 αποτελέσματα	
	R-tree	NO INDEX	R-tree	NO INDEX
TIME IN ms	74229	411536	30186	380679
MAP INPUT IN bytes	273	1188630739	273	1188630739
HDFS BYTES READ	5146043	1188700388	140120	1188700388

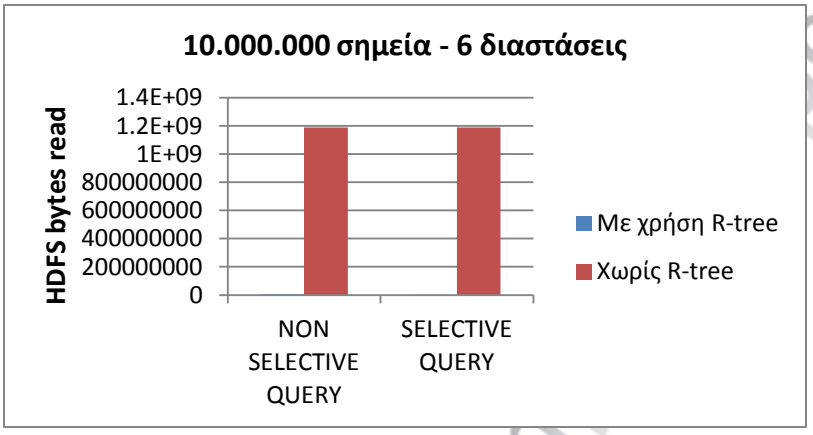
Ακολουθούν τα συγκριτικά διαγράμματα για το χρόνο εκτέλεσης της επερώτησης, τα map input bytes και τα bytes που διαβάστηκαν από το HDFS για σημεία 6 διαστάσεων.



Εικόνα 45: Χρόνος εκτέλεσης επερώτησης σε αρχείο 10.000.000 σημείων 6 διαστάσεων



Εικόνα 46: Bytes εισόδου στη συνάρτηση map για αρχείο 10.000.000 σημείων 6 διαστάσεων



Εικόνα 47: Bytes που διαβάστηκαν από το HDFS για αρχείο 10.000.000 σημείων 6 διαστάσεων

9. ΣΥΜΠΕΡΑΣΜΑΤΑ

Στην παρούσα εργασία αρχικά παρουσιάστηκε το Hadoop και το MapReduce και αναλύθηκε η αρχιτεκτονική και ο τρόπος λειτουργίας τους. Αναφέρθηκαν επίσης κάποια μειονεκτήματά τους που έχουν παρατηρηθεί και στη συνέχεια έγινε αναφορά σε μελέτες και προσπάθειες βελτίωσής τους. Επίσης παρουσιάστηκε η δομή ευρετηρίου του R-tree και περιγράφηκε ο τρόπος εκτέλεσης επερωτήσεων εύρους και κοντινότερου γείτονα με αξιοποίηση του ευρετηρίου.

Ακόλουθα παρουσιάστηκε το πρόβλημα στο οποίο η εργασία καλείται να απαντήσει, τέθηκαν οι στόχοι της και αναλύθηκε η σχεδίαση που ακολουθήθηκε καθώς και η υλοποίηση με αναφορά στον κώδικα που αναπτύχθηκε. Επόμενο βήμα ήταν να παρουσιαστεί η πειραματική μελέτη που πραγματοποιήθηκε με συγκριτικούς πίνακες και γραφήματα που αναπαριστούν τις μετρήσεις που έγιναν. Τα συμπεράσματα που προέκυψαν παρουσιάζονται ακόλουθα.

Οι μετρήσεις αποδεικνύουν ότι η χρήση ευρετηρίου όντως βελτιώνει την απόδοση του Hadoop στην εκτέλεση επερωτήσεων σε πολυδιάστατα δεδομένα. Μειώνει το χρόνο εκτέλεσης της επερώτησης, μειώνει σημαντικά την είσοδο της συνάρτησης map καθώς και τον αριθμό των bytes που διαβάζονται από το HDFS καθώς δεν απαιτείται πρόσβαση στο σύνολο των δεδομένων. Η βελτίωση είναι μεγαλύτερη όσο μεγαλύτερο είναι το μέγεθος του αρχείου δεδομένων και όσο πιο επιλεκτική είναι η επερώτηση που εκτελείται.

Τα μεγέθη που επιλέχθηκαν να μετρηθούν είναι ο χρόνος εκτέλεσης της επερώτησης, το input της συνάρτησης map καθώς και τα bytes που διαβάζονται από το HDFS. Οι μετρήσεις εκτελέστηκαν για αρχεία 1.000,10.000,100.000,1.000.000 και 10.000.000 σημείων με 4 και 6 διαστάσεις πάνω στα οποία εκτελέστηκαν επερωτήσεις εύρους με χρήση R-tree ευρετηρίου και χωρίς χρήση ευρετηρίου. Για κάθε περίπτωση εκτελέστηκαν δύο επερωτήσεις με διαφορετικό βαθμό επιλεκτικότητας.

Σε όλες τις περιπτώσεις ο χρόνος εκτέλεσης της επερώτησης είναι μικρότερος όταν χρησιμοποιείται R-tree ευρετήριο ενώ πολύ λιγότερα είναι και τα bytes που είναι το input της συνάρτησης map. Συγκεκριμένα στην περίπτωση χρήσης του ευρετηρίου το input της συνάρτησης map είναι μερικά bytes όσο και το μέγεθος του αρχείου μεταδεδομένων του ευρετηρίου. Αντίστοιχα όταν δεν χρησιμοποιείται ευρετήριο το input της συνάρτησης map είναι

ολόκληρο το αρχείο προς επεξεργασία. Επομένως η διαφορά είναι μεγαλύτερη όσο αυξάνεται ο αριθμός των σημείων και οι διαστάσεις τους.

Όσον αφορά τα bytes που διαβάζονται από το HDFS είναι σε όλες τις περιπτώσεις λιγότερα όταν χρησιμοποιείται ευρετήριο. Η διαφορά γίνεται ακόμα μεγαλύτερη όσο πιο επιλεκτική γίνεται η επερώτηση, δηλαδή όσο λιγότερα MBRs πρέπει να διαβαστούν. Στην περίπτωση που δε χρησιμοποιείται ευρετήριο τα bytes που διαβάζονται από το HDFS είναι της τάξης του μεγέθους του αρχείου προς επεξεργασία.

Σαν μελλοντική έρευνα προτείνεται η χρήση του κώδικα για την εκτέλεση και άλλων μορφών επερωτήσεων για να διαπιστωθεί αν και σε αυτές τις περιπτώσεις επιτυγχάνεται βελτίωση της απόδοσης. Επίσης προτείνεται να μελετηθεί η δυνατότητα να δημιουργείται το ευρετήριο του αρχείου κατά το ανέβασμα του στο HDFS και να μη χρειάζεται να δημιουργηθεί τοπικά.

Συνοπτικά, οι προτάσεις για βελτίωση και μελλοντική έρευνα είναι οι εξής:

- Επαναχρησιμοποίηση του κώδικα για εκτέλεση διαφορετικών τύπων επερωτήσεων
- Ενσωμάτωση της διαδικασίας δημιουργίας του ευρετηρίου του αρχείου στη διαδικασία ανεβάσματος στο HDFS
- Σύγκριση των αποτελεσμάτων της προσέγγισής μας με αντίστοιχες προγενέστερες έρευνες

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] Bo Dong, Qinghua Zheng, Jie Yang, Haifei Li, Mu Qiao: An E-learning Ecosystem Based on Cloud Computing Infrastructure. ICALT 2009: 125-127

[2] Daniel Warneke, Odej Kao: Exploiting Dynamic Resource Allocation for Efficient Parallel Data Processing in the Cloud. IEEE Trans. Parallel Distrib. Syst. 22(6): 985-997 (2011)

[3] Divyakant Agrawal, Sudipto Das, Amr El Abbadi: Big data and cloud computing: current state and future opportunities. EDBT 2011:530-533

[4] Shoji Nishimura, Sudipto Das, Divyakant Agrawal, Amr El Abbadi: MD-HBase: A Scalable Multi-dimensional Data Infrastructure for Location Aware Services. Mobile Data Management (1) 2011: 7-16

[5] S.Wu, D.Jiang, B.C.Ooi, K-L.Wu: Efficient B-tree based indexing for cloud data processing. PVLDB, 3(1), 2010.

[6] Kyong-Ha Lee, Yoon-Joon Lee, Hyunsik Choi, Yon Dohn Chung, Bongki Moon: Parallel data processing with MapReduce: a survey. SIGMOD Record 40(4): 11-20 (2011)

[7] K. Nørvåg, C. Doulkeridis: A Survey of Large-Scale Analytical Query Processing in MapReduce. *VLDB Journal,* (to appear)2013.

[8] J.Dean and S.Ghemawat. Mapreduce: Simplified data processing on large clusters. In OSDI 2004.

[9] J. Dean and S. Ghemawat. Mapreduce: a flexible data processing tool. Commun. ACM, 53(1):72–77, 2010.

[10]T. White. Hadoop: The Definitive Guide. O'Reilly Media, 3rd edition, June 2009.

[11] L. Chuck Hadoop in Action. Manning Publication Co, 2001.

[12]Jörg Schad, Jens Dittrich, Jorge-Arnulfo Quiané-Ruiz: Runtime Measurements in the Cloud: Observing, Analyzing, and Reducing Variance.PVLDB 3(1): 460-471 (2010)

[13]M.Stonebraker, D.J. Abadi, D.J. DeWitt, S.Madden, E.Paulson, A.Pavlo, A.Rasin: MapReduce and parallel DBMSs: friends or foes? Commun. ACM 53(1): 64-71 (2010).

[14]Dawei Jiang, Beng Chin Ooi, Lei Shi, Sai Wu. The Performance of MapReduce: An In-depth Study. PVLDB 3(1): 472-483 (2010)

[15]Jens Dittrich, Jorge-Arnulfo Quiané-Ruiz: Efficient Big Data Processing in Hadoop MapReduce. PVLDB 5(12): 2014-2015 (2012)

[16]Daniel J. Abadi: Data Management in the Cloud: Limitations and Opportunities. IEEE Data Eng. Bull. 32(1): 3-12 (2009)

[17] B.F.Cooper, R.Ramakrishnan, U.Srivastava, A.Silberstein, P.Bohannon, H.A.Jacobsen, N.Puz, D.Weaver, and R.Yerneni. Pnuts: Yahoo!'s hosted data serving platform. In VLDB 2008.

[18] A. Thusoo, J.S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H.Liu, P. Wyckoff, R.Murthy: Hive - A warehousing solution over a map-reduce framework. PVLDB 2(2): 1626-1629 (2009).

[19]Jens Dittrich, Stefan Richter, Stefan Schuh: Efficient OR Hadoop: Why Not Both? Datenbank-Spektrum 13(1): 17-22 (2013)

[20]«Apache Hadoop», [Ηλεκτρονικό]. Available: http://hadoop.apache.org

[21] «Hadoop Wiki», [Ηλεκτρονικό]. Available: http://wiki.apache.org/hadoop/

[22] [Ηλεκτρονικό]. Available: http://www.rohitmenon.com/index.php/category/hadoop/

[23]Christos Doulkeridis, Kjetil Nørvåg: On saying "enough already!" in MapReduce. Cloud-I 2012: 7

[24]Azza Abouzeid, Kamil Bajda-Pawlikowski, Daniel J. Abadi, Alexander Rasin, Avi Silberschatz: HadoopDB: An Architectural Hybrid of MapReduce and DBMS Technologies for Analytical Workloads. PVLDB 2(1): 922-933 (2009)

[25]Jens Dittrich, Jorge-Arnulfo Quiané-Ruiz, Alekh Jindal, Yagiz Kargin, Vinay Setty, Jörg Schad: Hadoop++: Making a Yellow Elephant Run Like a Cheetah (Without It Even Noticing). PVLDB 3(1): 518-529 (2010)

[26] Jens Dittrich, Jorge-Arnulfo Quiané-Ruiz, Stefan Richter, Stefan Schuh, Alekh Jindal, Jörg Schad: Only Aggressive Elephants are Fast Elephants.PVLDB 5(11): 1591-1602 (2012)

[27] X.Zhang, J.Ai, Z. Wang, J.Lu, and X.Meng: An efficient multi-dimensional index for cloud data management. In CloudDB'09: 17-24.

[28] J.Wang, S.Wu, H.Gao, J.Li, B.C.Ooi: Indexing multi-dimensional data in a cloud system. SIGMOD Conference 2010: 591-602.

[29] Aleksandar Stupar, Sebastian Michel, Ralf Schenkel: RankReduce - processing K-Nearest Neighbor Queries on Top of MapReduce. LSDS-IR 2010

[30] Ahmed Eldawy, Mohamed F. Mokbel: A Demonstration of SpatialHadoop: An Efficient MapReduce Framework for Spatial Data. PVLDB 6(12): 1230-1233 (2013)

[31]Hoang Tam Vo, Chun Chen, Beng Chin Ooi: Towards Elastic Transactional Cloud Storage with Range Query Support. PVLDB 3(1): 506-517 (2010)

[32]R. Chaiken, B. Jenkins, P.-A. Larson, B. Ramsey, D. Shakib, S. Weaver, and J. Zhou. Scope: easy and efficient parallel processing of massive data sets. Proc. VLDB Endow., 1(2):1265–1276, 2008.

[33] Mohamed Y. Eltabakh, Yuanyuan Tian, Fatma Özcan, Rainer Gemulla, Aljoscha Krettek, John McPherson: CoHadoop: Flexible Data Placement and Its Exploitation in Hadoop. PVLDB 4(9): 575-585 (2011)

[34]«Wikipedia – The free encyclopedia», [Ηλεκτρονικό]. Available: http://www.wikipedia.org

[35] Antonin Guttman: R-Trees: A Dynamic Index Structure for Spatial Searching. SIGMOD Conference 1984: 47-57

[36] Apostolos Papadopoulos, Yannis Manolopoulos: Performance of Nearest Neighbor Queries in R-Trees. ICDT 1997: 394-408

[37] Christos Doulkeridis, Akrivi Vlachou, Yannis Kotidis, Michalis Vazirgiannis: Efficient range query processing in metric spaces over highly distributed data. Distributed and Parallel Databases 26(2-3): 155-180 (2009)

[38]«Eclipse» The Eclipse Foundation, [Ηλεκτρονικό]. Available: http://www.eclipse.org. [Πρόσβαση 2011].

[39]N.Mamoulis.Spatial Data Management.Morgan & Claypool Publishers

ΠΑΡΑΡΤΗΜΑ

WholeFileInputFormat

```
public static class WholeFileInputFormat extends FileInputFormat<LongWritable,
Text> {
      @Override
      protected boolean isSplitable(FileSystem fs, Path filename) {
          return false;
      }

      @Override
      public RecordReader<LongWritable, Text> getRecordReader(
InputSplit split, JobConf job, Reporter reporter) throws IOException {
              return new WholeFileRecordReader((FileSplit) split,job);
          }
      }
```

WholeFileRecordReader

```
 public static class WholeFileRecordReader implements
 RecordReader<LongWritable, Text> {

         private FileSplit fileSplit;
         private Configuration conf;
         private boolean processed = false;


public WholeFileRecordReader(FileSplit split, Configuration conf) throws
IOException {
             this.fileSplit = split;
             this.conf = conf;
}

@Override
public boolean next(LongWritable key, Text value) throws IOException {
             if (!processed) {
                 Path file = fileSplit.getPath();
                 String fileName = file.getName();
                 key.set(1);
                 FSDataInputStream in = null;
                 try {
                     FileSystem dfs = FileSystem.get(conf);
                     in = dfs.open(file);
                     ObjectInputStream os1 = new ObjectInputStream(in);
                     LeafMetaData metaData = (LeafMetaData)os1.readObject();
                     LoadTree datatree = new LoadTree(fileName.substring(0,
                     fileName.indexOf('.')), metaData,dfs);
                     int targetLevel = 0; //THIS SHOULD BE ALWAYS 0
                    Cursor cursor = datatree.rtree.query(query,targetLevel);
```

```
                    //Anaktoume ena-ena ta apotelesmata

                      StringBuffer s = new StringBuffer();
                      while (cursor.hasNext())
                      {
                            KPE nt = (KPE)cursor.next();
                            DoublePoint smallPoint =
                            DoublePoint)((DoublePointRectangle)nt.getData()
                            ).getCorner(true);
                            s.append(smallPoint.toString());
                            s.append("\n");

                      }
                      value.set(new Text(s.toString()));
                } catch (ClassNotFoundException e) {
                            // TODO Auto-generated catch block
                            e.printStackTrace();
                      } finally {
                    IOUtils.closeStream(in);
                }
                processed = true;
                return true;
            }

              return false;
        }

          @Override
          public void close() throws IOException {
                // TODO Auto-generated method stub

          }
          @Override
          public LongWritable createKey() {
                // TODO Auto-generated method stub
                return new LongWritable();
          }
          @Override
          public Text createValue() {
                // TODO Auto-generated method stub
                return new Text();
          }
          @Override
          public long getPos() throws IOException {
                // TODO Auto-generated method stub
                return processed ? fileSplit.getLength() : 0;
          }
          @Override
          public float getProgress() throws IOException {
                // TODO Auto-generated method stub
                return processed ? 1.0f : 0.0f;
          }

    }
```

Map συνάρτηση για εκτέλεση επερώτησης χρήση ευρετηρίου

```
public static class Map extends
MapReduceBase implements
Mapper<LongWritable, Text, Text,
IntWritable> {
      private Text word = new Text();
      public void map(LongWritable key,
                  Text value, OutputCollector<Text,
                  IntWritable> output, Reporter
                  reporter) throws IOException {
               BufferedReader reader = new BufferedReader(new
               StringReader(value.toString()));
               String line;
               int i = 0;
               while((line = reader.readLine())!=null){
               i++;
               output.collect(new Text(line), new IntWritable(i));
   }

      }
}
```

Reduce συνάρτηση για εκτέλεση επερώτησης χρήση ευρετηρίου

```
public static class Reduce extends
MapReduceBase implements
Reducer<Text, IntWritable, Text,
IntWritable> {
      public void reduce(Text key,
                  Iterator<IntWritable> values,
                  OutputCollector<Text, IntWritable>
                  output, Reporter reporter) throws
                  IOException {
            output.collect(key, new IntWritable(1));

      }
}
```

Map συνάρτηση για εκτέλεση επερώτησης χωρίς χρήση ευρετηρίου

```
public static class MapNoIndex extends
  MapReduceBase implements
  Mapper<LongWritable, Text, Text,
  IntWritable> {
  private final static IntWritable
  one = new IntWritable(1);
  private Text word = new Text();
  public void map(LongWritable key,
                  Text value, OutputCollector<Text,
                  IntWritable> output, Reporter
                  reporter) throws IOException {
      DoublePointRectangle query = new DoublePointRectangle(coor1,coor2);
      double dCoord[] = new double[dim];
      StringTokenizer st = new StringTokenizer(value.toString()," \t");
      Long id=Long.parseLong(st.nextToken());
      for (int j=0;j<dim;j++)
        {
         dCoord[j] = Double.parseDouble(st.nextToken());
        }
        DoublePoint d = new DoublePoint(dCoord);
        if(query.contains(d)){
              word.set(d.toString());
              output.collect(word, one);
        }
        }
        }
```

Reduce συνάρτηση για εκτέλεση επερώτησης χωρίς χρήση ευρετηρίου

```
public static class Reduce extends
   MapReduceBase implements
   Reducer<Text, IntWritable, Text,
   IntWritable> {
         public void reduce(Text key,
                     Iterator<IntWritable> values,
                     OutputCollector<Text, IntWritable>
                     output, Reporter reporter) throws
                     IOException {
               output.collect(key, new IntWritable(1));

         }
   }
```